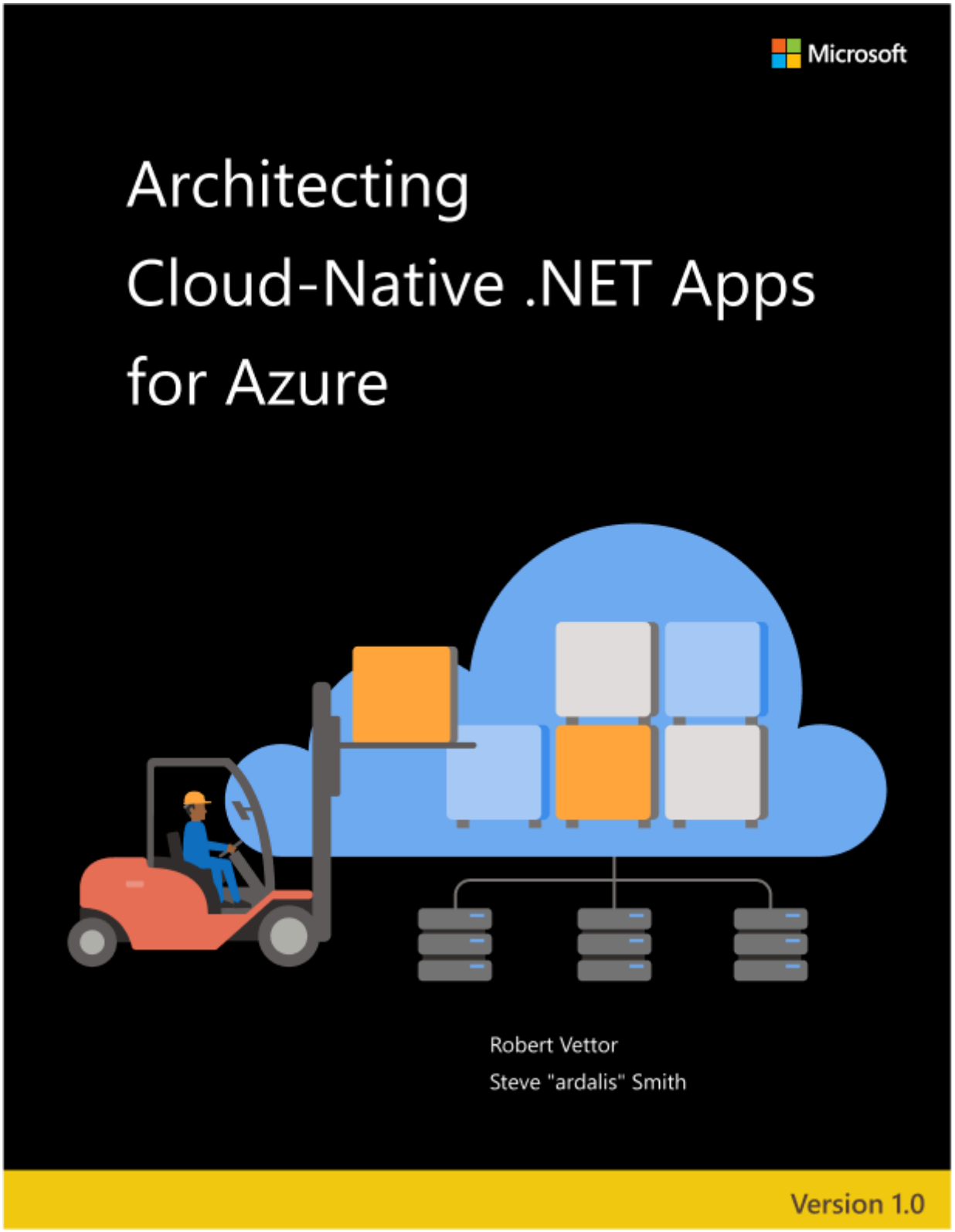
Díganos qué opina sobre la experiencia de descarga del PDF.

Diseño de aplicaciones .NET nativas en la nube para Azure

18/06/2025



EDITION v1.0.3

Consulte el registro de cambios para obtener las actualizaciones del libro y las contribuciones de la comunidad.

PUBLICADO POR

Equipos de producto de Microsoft Developer Division, .NET y Visual Studio

Una división de Microsoft Corporation

One Microsoft Way

Redmond, Washington 98052-6399

Copyright © 2023 de Microsoft Corporation

Todos los derechos reservados. No se podrá reproducir ni transmitir ninguna parte del contenido de este libro en cualquier forma ni por ningún medio sin el permiso por escrito del editor.

Este libro se proporciona “tal cual” y expresa las opiniones del autor. Las vistas, opiniones e información expresadas en este libro, incluidas las direcciones URL y otras referencias a sitios web de Internet, pueden cambiar sin previo aviso.

Algunos ejemplos que se muestran aquí se proporcionan solo para la ilustración y son ficticios. No se pretende ni debe deducirse ninguna asociación o conexión real.

Microsoft y las marcas registradas que aparecen en https://www.microsoft.com la página web "Marcas comerciales" son marcas comerciales del grupo de empresas de Microsoft.

Mac y macOS son marcas comerciales de Apple Inc.

El logotipo de la ballena de Docker es una marca registrada de Docker, Inc. Usado por permiso.

Todas las demás marcas y logotipos son propiedad de sus respectivos propietarios.

Autores:

> **Rob Vettor**, Principal MTC (Microsoft Technology Center) Architect for Cloud App Innovation, Microsoft
>
> **Steve "ardalis" Smith**, arquitecto de software y entrenador - Ardalis.com

Participantes y revisores:

> **Cesar De la Torre**, director de programas principal, equipo de .NET, Microsoft
>
> **Nish Anil**, jefe de programas sénior, equipo de .NET, Microsoft
>
> **Jeremy Likness**, jefe de programas sénior, equipo de .NET, Microsoft

> **Cecil Phillip**, promotor sénior de la nube, Microsoft
>
> **Sumit Ghosh**, consultor principal de Neudesic

Editores:

> **Maira Wenzel**, Administrador de programas, equipo de .NET, Microsoft

> **David Pine**, desarrollador sénior de contenido, documentos de .NET, Microsoft

Versión

Esta guía se ha escrito para cubrir **la versión de .NET 7** junto con muchas actualizaciones adicionales relacionadas con la misma "oleada" de tecnologías (es decir, Azure y tecnologías de terceros adicionales) que coinciden en el tiempo con la versión de .NET 7.

Quién debe usar esta guía

El público de esta guía es principalmente desarrolladores, responsables de desarrollo y arquitectos interesados en aprender a crear aplicaciones diseñadas para la nube.

Una audiencia secundaria es responsables de la toma de decisiones técnicas que planean elegir si crear sus aplicaciones mediante un enfoque nativo de la nube.

Cómo puede usar esta guía

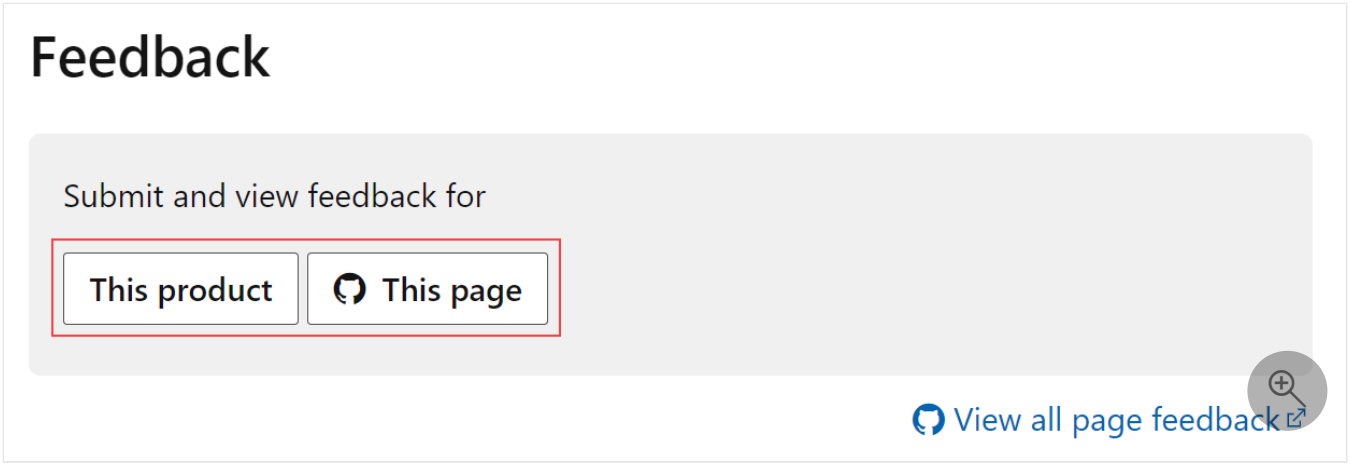
Esta guía comienza definiendo la nube nativa e introduciendo una aplicación de referencia creada mediante principios y tecnologías nativos de la nube. Más allá de estos dos primeros capítulos, el resto del libro se divide en capítulos específicos centrados en temas comunes a la mayoría de las aplicaciones nativas de la nube. Puede ir a cualquiera de estos capítulos para obtener información sobre los enfoques nativos de la nube para:

- Datos y acceso a datos
- Patrones de comunicación
- Escalado y escalabilidad
- Resistencia de la aplicación
- Supervisión y mantenimiento
- Identidad y seguridad
- DevOps

Esta guía está disponible en formato PDF y en línea. No dude en reenviar este documento o vínculos a su versión en línea al equipo para ayudar a garantizar la comprensión común de estos temas. La mayoría de estos temas se benefician de una comprensión coherente de los principios y patrones subyacentes, así como de los inconvenientes implicados en las decisiones relacionadas con estos temas. Nuestro objetivo con este documento es equipar a los equipos y a sus líderes con la información que necesitan para tomar decisiones bien fundamentadas para la arquitectura, el desarrollo y el hospedaje de sus aplicaciones.

Enviar sus comentarios

Este libro y ejemplos relacionados evolucionan constantemente, por lo que se agradecen sus comentarios. Si tiene comentarios sobre cómo se puede mejorar este libro y está leyendo este libro en <learn.microsoft.com>, use la sección **Comentarios** en la parte inferior de la página:



Como se ha resaltado en la captura de pantalla anterior, la sección de comentarios le permite enviar comentarios para:

- **Este producto**: mediante el formulario de comentarios del producto .NET.
- **Esta página**: usando una plantilla de incidencia de GitHub con los detalles de la página.

Si está leyendo este libro como un PDF, puede enviar comentarios creando un problema de GitHub en Documentos de .NET o mediante la plantilla de problemas de GitHub en Libros electrónicos de arquitectura de .NET.

Próximo

Introducción a las aplicaciones nativas de la nube

19/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Otro día más en la oficina, trabajando en "el próximo gran hito".

Tu teléfono celular suena. Es su amable reclutador - el que llama a diario con nuevas oportunidades emocionantes.

Pero esta vez es diferente: Empresa emergente, capital y abundante financiación.

La mención de la nube, los microservicios y la tecnología de vanguardia le llevan al límite.

Avance rápido unas semanas y ahora eres un nuevo empleado en una sesión de diseño que diseña una aplicación de comercio electrónico importante. Competirás con los principales sitios de comercio electrónico.

¿Cómo lo crearás?

Si sigue las instrucciones de los últimos 15 años, probablemente compilará el sistema que se muestra en la figura 1.1.

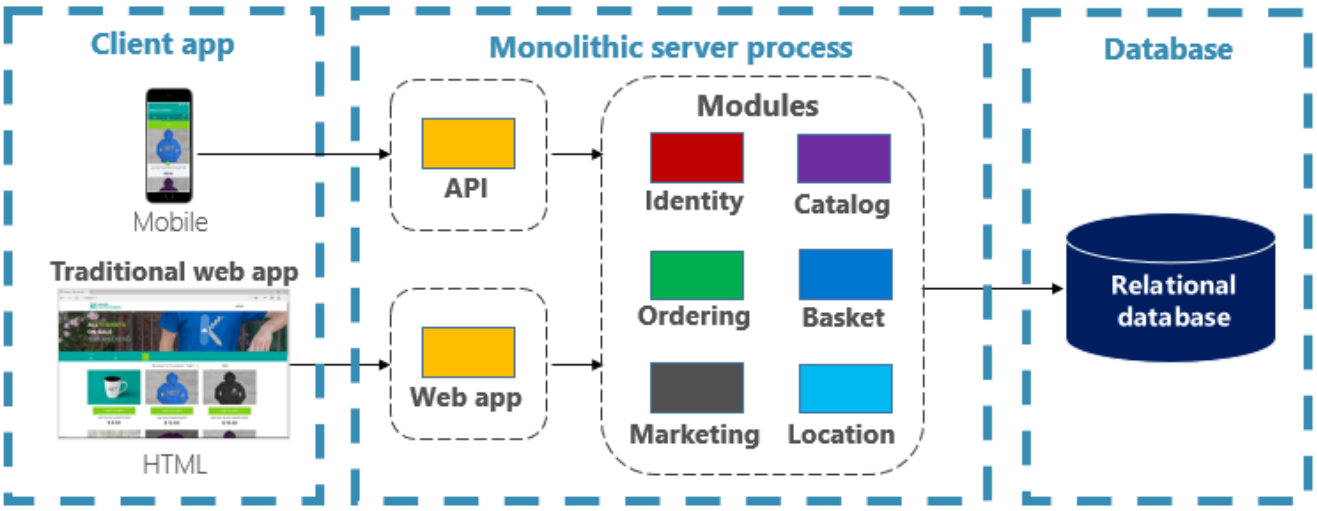


Figura 1-1. Diseño monolítico tradicional

Se crea una aplicación principal de gran tamaño que contiene toda la lógica de dominio. Incluye módulos como Identity, Catalog, Ordering, etc. Se comunican directamente entre sí dentro de un único proceso de servidor. Los módulos comparten una base de datos relacional grande. El núcleo expone la funcionalidad a través de una interfaz HTML y una aplicación móvil.

¡Felicidades! Acabas de crear una aplicación monolítica.

No todo es malo. Los monolitos ofrecen algunas ventajas distintas. Por ejemplo, son sencillos de...

- Construir
- prueba
- desplegar
- solucionar problemas
- escalar verticalmente

Muchas aplicaciones exitosas que existen hoy en día fueron creadas como monolitos. La aplicación es un éxito y continúa evolucionando, iteración después de la iteración, agregando más funcionalidad.

Sin embargo, en algún momento empiezas a sentirte incómodo. Está perdiendo control sobre la aplicación. A medida que pasa el tiempo, la sensación se vuelve más intensa y finalmente entras en un estado conocido como `Fear Cycle`:

- La aplicación se ha vuelto tan abrumadoramente complicada que nadie lo entiende.
- Usted teme realizar cambios: cada cambio tiene efectos secundarios no deseados y costosos.
- Las nuevas características y correcciones se vuelven complicadas, lentas y costosas de implementar.

- Cada versión se vuelve lo más pequeña posible y requiere una implementación completa de toda la aplicación.
- Un componente inestable puede bloquear todo el sistema.
- Las nuevas tecnologías y marcos no son una opción.
- Es difícil implementar metodologías de entrega ágiles.
- La erosión arquitectónica se instala a medida que la base de código se deteriora con interminables "correcciones en el momento".
- Por último, los *consultores* entran y le dicen que lo reescriba.

¿Suena familiar?

Muchas organizaciones han abordado este ciclo de miedo monolítico mediante la adopción de un enfoque nativo de la nube para crear sistemas. En la figura 1-2 se muestra el mismo sistema creado para aplicar técnicas y prácticas nativas de la nube.

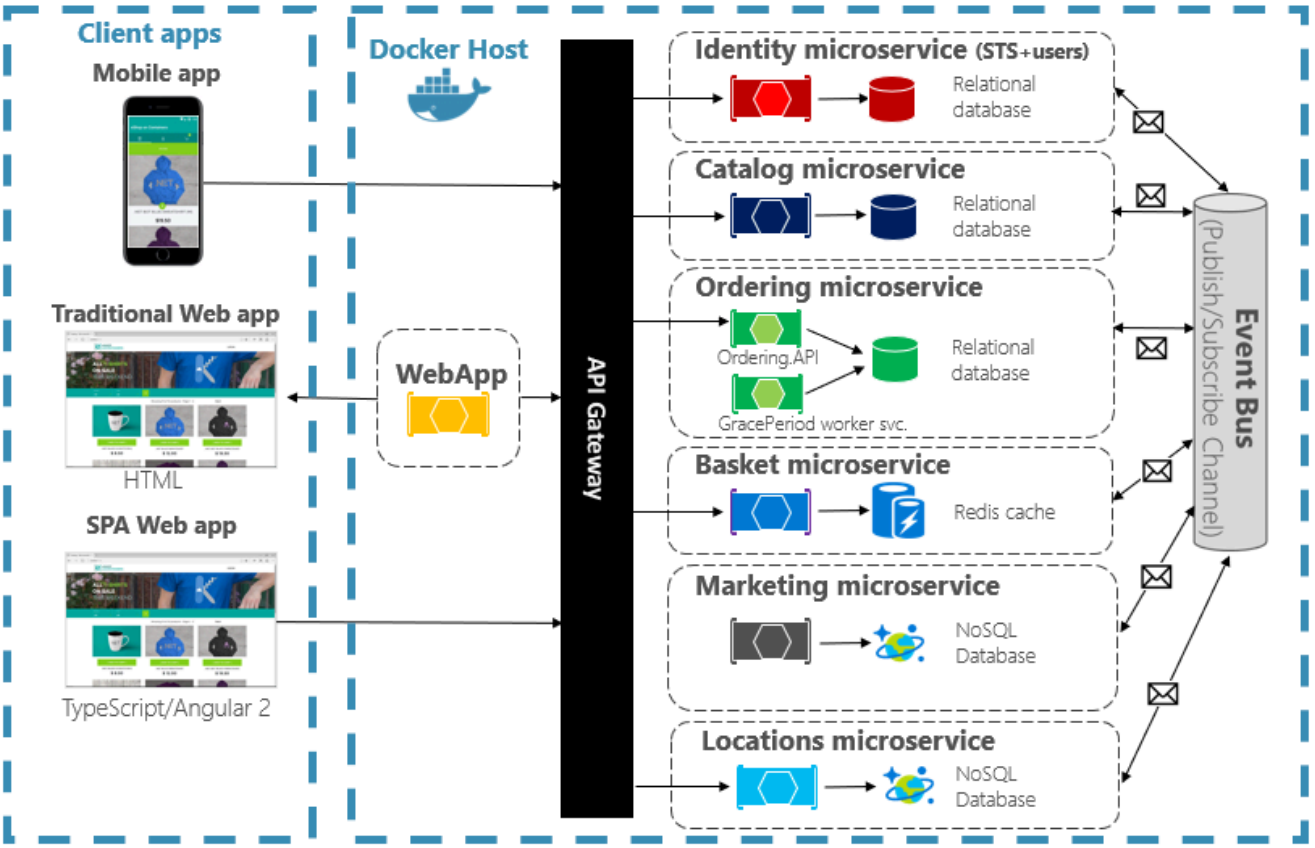


Figura 1-2. Diseño nativo de la nube

Observe cómo se descompone la aplicación en un conjunto de microservicios aislados pequeños. Cada servicio es independiente y encapsula su propio código, datos y dependencias. Cada uno se implementa en un contenedor de software y lo administra un orquestador de contenedores. En lugar de una base de datos relacional grande, cada servicio posee su propio almacén de datos, el tipo de que varía en función de las necesidades de datos. Tenga en cuenta cómo algunos servicios dependen de una base de datos relacional, pero otras de bases de datos NoSQL. Un servicio almacena su estado en una caché distribuida. Tenga en

cuenta cómo todo el tráfico se canaliza a través de un servicio de API Gateway, el cual es responsable de dirigir el tráfico hacia los servicios centrales del back-end y de aplicar muchos temas transversales. Lo más importante es que la aplicación aproveche al máximo las características de escalabilidad, disponibilidad y resistencia que se encuentran en las plataformas en la nube modernas.

Informática nativa en la nube

Hmm... Acabamos de usar el término *Cloud Native*. Tu primer pensamiento podría ser: "¿Qué significa exactamente eso?" ¿Otra palabra de moda del sector codificada por los proveedores de software para comercializar más cosas?"

Afortunadamente es muy diferente, y esperamos que este libro le ayude a convencerle.

En poco tiempo, el nativo de la nube se ha convertido en una tendencia impulsora en el sector del software. Es una nueva manera de construir sistemas grandes y complejos. El enfoque aprovecha al máximo las prácticas modernas de desarrollo de software, las tecnologías y la infraestructura en la nube. La nube nativa cambia la forma de diseñar, implementar, implementar y poner en funcionamiento sistemas.

A diferencia de las constantes expectativas que caracterizan a nuestro sector, lo nativo de nube es *real*. Considere la Cloud Native Computing Foundation (CNCF), un consorcio de más de 400 grandes corporaciones. Su misión es hacer que la informática nativa de la nube sea omnipresente en toda la tecnología y las soluciones en la nube. Como uno de los grupos de código abierto más influyentes, hospeda muchos de los proyectos de código abierto de mayor crecimiento en GitHub. Estos proyectos incluyen Kubernetes, Prometheus, Helm, Envoy y gRPC.

El CNCF fomenta un ecosistema de neutralidad de código abierto y proveedor. Siguiendo esa dirección, este libro presenta principios, patrones y mejores prácticas diseñados para la nube que son independientes de la tecnología. Al mismo tiempo, se describen los servicios y la infraestructura disponibles en la nube de Microsoft Azure para construir sistemas nativos de nube.

Entonces, ¿qué es exactamente Cloud Native? Siéntate, relájate y déjanos ayudarte a explorar este nuevo mundo.

Siguiente anterior

¿Qué es Cloud Native?

17/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Detenga lo que está haciendo y pida a sus compañeros que definan el término "Cloud Native". Hay una buena oportunidad de obtener varias respuestas diferentes.

Comencemos con una definición sencilla:

> *La arquitectura y las tecnologías nativas de la nube son un enfoque para diseñar, construir y operar cargas de trabajo integradas en la nube y aprovechar al máximo el modelo de informática en la nube.*

Cloud Native Computing Foundation proporciona la definición oficial:

> *Las tecnologías nativas de la nube permiten a las organizaciones crear y ejecutar aplicaciones escalables en entornos modernos y dinámicos, como nubes públicas, privadas e híbridas. Los contenedores, las mallas de servicio, los microservicios, la infraestructura inmutable y las API declarativas ejemplifican este enfoque.*

> *Estas técnicas permiten sistemas acoplados flexiblemente resistentes, manejables y observables. Junto con una automatización sólida, permiten a los ingenieros realizar cambios de alto impacto con frecuencia y predicción con un trabajo mínimo.*

La nube nativa consiste en la *velocidad* y *la agilidad*. Los sistemas empresariales evolucionan desde la habilitación de capacidades empresariales a armas de transformación estratégica que aceleran la velocidad y el crecimiento del negocio. Es imperativo conseguir nuevas ideas para comercializar inmediatamente.

Al mismo tiempo, los sistemas empresariales también se han vuelto cada vez más complejos con los usuarios que exigen más. Esperan una capacidad de respuesta rápida, características innovadoras y un tiempo de inactividad cero. Los problemas de rendimiento, los errores recurrentes y la incapacidad de mover rápidamente ya no son aceptables. que harán que los usuarios empiecen a pensar en irse a la competencia. Los sistemas nativos en la nube están diseñados para adoptar cambios rápidos, grandes escalas y resistencia.

Estas son algunas empresas que han implementado técnicas nativas de nube. Piense en la velocidad, agilidad y escalabilidad que han logrado.

Expandir tabla

Empresa	Experiencia
Netflix	Tiene más de 600 servicios en producción. Realiza implementaciones 100 veces al día.
Uber	Tiene más de 1000 servicios en producción. Despliega varios miles de veces cada semana.
WeChat	Tiene más de 3000 servicios en producción. Despliega 1,000 veces al día.

Como puede ver, Netflix, Uber y WeChat exponen sistemas nativos en la nube que constan de muchos servicios independientes. Este estilo arquitectónico les permite responder rápidamente a las condiciones de mercado. Actualizan instantáneamente áreas pequeñas de una aplicación activa y compleja, sin una reimplementación completa. Escalan individualmente los servicios según sea necesario.

Los pilares de los elementos nativos de la nube

La velocidad y agilidad de la nube nativa derivan de muchos factores. Lo más importante es la *infraestructura en la nube*. Pero hay más: Otros cinco pilares fundamentales que se muestran en la figura 1-3 también proporcionan la base para los sistemas nativos de la nube.

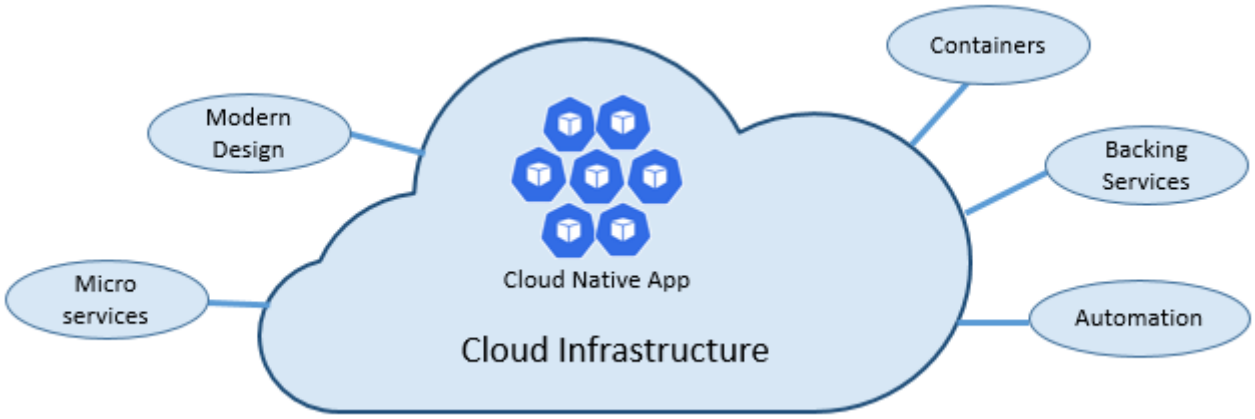


Figura 1-3. Pilares fundamentales nativos de la nube

Dediquemos un tiempo a comprender mejor la importancia de cada pilar.

La nube

Los sistemas nativos en la nube aprovechan al máximo el modelo de servicio en la nube.

Diseñado para prosperar en un entorno dinámico y virtualizado en la nube, estos sistemas hacen un uso amplio de la infraestructura de proceso de plataforma como servicio (PaaS) y los servicios administrados. Tratan la infraestructura subyacente como *descartable* (aprovisionada en minutos y cambiada de tamaño, escalada o destruida a petición) a través de la automatización.

Tenga en cuenta la diferencia entre cómo tratamos las mascotas y los productos básicos. En un centro de datos tradicional, los servidores se tratan como mascotas: una máquina física, con un nombre significativo y cuidado. Para escalarlos, agregue más recursos a la misma máquina (escalado vertical). Si el servidor se enferma, lo cuidas hasta que recupere la salud. Si el servidor se vuelve inaccesible, todos se dan cuenta.

El modelo de servicio de productos básicos es diferente. Aprovisiona cada instancia como una máquina virtual o un contenedor. Son idénticos y asignados a un identificador del sistema, como Service-01, Service-02, etc. Puede escalar mediante la creación de más instancias (escalado horizontal). Nadie observa cuándo una instancia deja de estar disponible.

El modelo de productos básicos adopta una infraestructura inmutable. Los servidores no se reparan ni modifican. Si se produce un error o se requiere la actualización, se destruye y se aprovisiona uno nuevo, todo ello a través de la automatización.

Los sistemas nativos en la nube adoptan el modelo de servicio de productos básicos. Siguen ejecutándose cuando la infraestructura se escala o se reduce horizontalmente, independientemente de las máquinas en las que se ejecutan.

La plataforma en la nube de Azure admite este tipo de infraestructura altamente elástica con funcionalidades de escalado automático, recuperación automática y supervisión.

Diseño moderno

¿Cómo diseñaría una aplicación nativa de nube? ¿Qué aspecto tendría la arquitectura? ¿A qué principios, patrones y procedimientos recomendados se adhiere? ¿Qué problemas operativos y de infraestructura serían importantes?

La aplicación Twelve-Factor

Una metodología ampliamente aceptada para construir aplicaciones basadas en la nube es la aplicaciónTwelve-Factor. Describe un conjunto de principios y prácticas que siguen los desarrolladores para construir aplicaciones optimizadas para entornos de nube modernos. Se presta especial atención a la portabilidad entre entornos y automatización declarativa.

Aunque es aplicable a cualquier aplicación basada en web, muchos profesionales consideran Twelve-Factor una base sólida para crear aplicaciones nativas de la nube. Los sistemas creados sobre estos principios pueden implementar y escalar rápidamente y agregar características para reaccionar rápidamente a los cambios de mercado.

En la tabla siguiente se resalta la metodología de Twelve-Factor:

Expandir tabla

Factor	Explicación
1 - Base de código	Una base de código única para cada microservicio, almacenada en su propio repositorio. Se realiza un seguimiento con el control de versiones, se puede implementar en varios entornos (QA, Ensayo, Producción).
2- Dependencias	Cada microservicio aísla y empaqueta sus propias dependencias, adoptando cambios sin afectar a todo el sistema.
3 - Configuraciones	La información de configuración se mueve fuera del microservicio y se externaliza a través de una herramienta de administración de configuración fuera del código. La misma implementación se puede propagar entre entornos con la configuración correcta aplicada.
4 - Servicios de respaldo	Los recursos auxiliares (almacenes de datos, cachés, agentes de mensajes) deben exponerse a través de una dirección URL direccionable. Si lo hace, desacopla el recurso de la aplicación, lo que le permite ser intercambiable.
5- Compilación, versión, ejecución	Cada versión debe aplicar una separación estricta entre las fases de compilación, versión y ejecución. Cada uno debe etiquetarse con un identificador único y

Factor	Explicación
	permitir la reversión. Los sistemas de CI/CD modernos ayudan a cumplir este principio.
6 - Procesos	Cada microservicio debe ejecutarse en su propio proceso, aislado de otros servicios en ejecución. Externalice el estado necesario a un servicio de respaldo, como una caché distribuida o un almacén de datos.
7: Enlace a puerto	Cada microservicio debe estar independiente con sus interfaces y funcionalidades expuestas en su propio puerto. Al hacerlo, se proporciona aislamiento de otros microservicios.
8: Simultaneidad	Cuando se necesite aumentar la capacidad, escale los servicios en sentido horizontal mediante varios procesos idénticos (copias) en lugar de escalar en sentido vertical una sola instancia grande sobre la máquina más potente disponible. Desarrolle la aplicación para que sea simultánea, lo que hace que el escalado horizontal en entornos en la nube sea sin problemas.
9 - Desechabilidad	Las instancias de servicio deben poder descartarse. Favorece el inicio rápido para aumentar las oportunidades de escalabilidad y el apagado elegante para dejar el sistema en un estado adecuado. Los contenedores de Docker junto con un orquestador cumplen inherentemente este requisito.
10 - Paridad entre Desarrollo y Producción	Mantenga los entornos en el ciclo de vida de la aplicación lo más parecido posible, lo que evita métodos abreviados costosos. Aquí, la adopción de contenedores puede contribuir considerablemente al promover el mismo entorno de ejecución.
11 - Registro	Trate los registros generados por microservicios como flujos de eventos. Procesarlos usando un agregador de eventos. Propague los datos de registro a herramientas de administración de registros o minería de datos, como Azure Monitor o Splunk y, finalmente, al archivo a largo plazo.
12 - Procesos de administración	Ejecute tareas administrativas o de administración, como limpieza de datos o análisis informáticos, como procesos puntuales. Use herramientas independientes para invocar estas tareas desde el entorno de producción, pero por separado de la aplicación.

En el libro Beyond the Twelve-Factor App, el autor Kevin Hoffman detalla cada uno de los 12 factores originales (escritos en 2011). Además, analiza tres factores adicionales que reflejan el diseño moderno de aplicaciones en la nube actuales.

Expandir tabla

Nuevo factor	Explicación
13 - API Primero	Convierte todo en un servicio. Supongamos que el código lo consumirá un cliente front-end, una puerta de enlace u otro servicio.

Nuevo factor	Explicación
14- Telemetría	En una estación de trabajo, tienes una visión profunda de tu aplicación y su comportamiento. Sin embargo, en la nube, no sucede lo mismo. Asegúrese de que su diseño incluya la recopilación de datos de supervisión, datos específicos del dominio y datos de salud o sistema.
15 - Autenticación y autorización	Implemente la identidad desde el principio. Considere las características de RBAC (control de acceso basado en rol) disponibles en nubes públicas.

Nos referiremos a muchos de los 12 factores de este capítulo y a lo largo del libro.

Marco de buena arquitectura de Azure

Diseñar e implementar cargas de trabajo basadas en la nube puede ser difícil, especialmente al implementar la arquitectura nativa de la nube. Microsoft proporciona procedimientos recomendados estándar del sector para ayudarle a usted y a su equipo a ofrecer soluciones sólidas en la nube.

Microsoft Well-Architected Framework proporciona un conjunto de principios rectores que se pueden usar para mejorar la calidad de una carga de trabajo nativa de la nube. El marco consta de cinco pilares de excelencia de la arquitectura:

Expandir tabla

Principios	Descripción
Gestión de costes	Céntrese en generar el valor incremental al principio. Aplique los principios *build-Measure-Learn* para acelerar el tiempo de comercialización, a la vez que evita soluciones de uso intensivo de capital. Utilice una estrategia de pago por uso para invertir a medida que escala horizontalmente, en lugar de ofrecer una gran inversión por adelantado.
Excelencia operativa	Automatice el entorno y las operaciones para aumentar la velocidad y reducir los errores humanos. Revierta las actualizaciones de problemas o póngalas al día rápidamente. Implemente la supervisión y el diagnóstico desde el principio.
Eficacia del rendimiento	Satisfaga de forma eficaz las demandas que se colocan en las cargas de trabajo. Favorezca el escalado horizontal y diséñelo en los sistemas. Realice continuamente pruebas de rendimiento y carga para identificar potenciales cuellos de botella.
Confiabilidad	Cree cargas de trabajo que sean resistentes y disponibles. La resistencia permite que las cargas de trabajo se recuperen de errores y sigan funcionando. La disponibilidad garantiza el acceso de los usuarios a la carga de trabajo en todo momento. Diseñe aplicaciones para esperar errores y recuperarse de ellas.

Principios	Descripción
Seguridad	Implemente la seguridad en todo el ciclo de vida de una aplicación, desde el diseño y la implementación hasta la implementación y las operaciones. Preste mucha atención a la administración de identidades, el acceso a la infraestructura, la seguridad de las aplicaciones y la soberanía y el cifrado de datos.

Para empezar, Microsoft proporciona un conjunto de evaluaciones en línea que le ayudarán a evaluar las cargas de trabajo actuales en la nube con los cinco pilares bien diseñados.

Microservicios

Los sistemas nativos de la nube adoptan microservicios, un estilo arquitectónico popular para construir aplicaciones modernas.

Creado como un conjunto distribuido de servicios pequeños e independientes que interactúan a través de un tejido compartido, los microservicios comparten las siguientes características:

- Cada uno implementa una funcionalidad empresarial específica dentro de un contexto de dominio mayor.
- Cada uno se desarrolla de forma autónoma y se puede implementar de forma independiente.
- Cada uno es autónomo y encapsula su propia tecnología de almacenamiento de datos, dependencias y plataforma de programación.
- Cada se ejecuta en su propio proceso y se comunica con otros usuarios mediante protocolos de comunicación estándar como HTTP/HTTPS, gRPC, WebSockets o AMQP.
- Se unen para formar una aplicación.

La figura 1-4 contrasta un enfoque de aplicación monolítica con un enfoque de microservicios. Observe cómo el monolito se compone de una arquitectura superpuesta, que se ejecuta en un único proceso. Normalmente consume una base de datos relacional. Sin embargo, el enfoque de microservicios separa la funcionalidad en servicios independientes, cada uno con su propia lógica, estado y datos. Cada microservicio hospeda su propio almacén de datos.

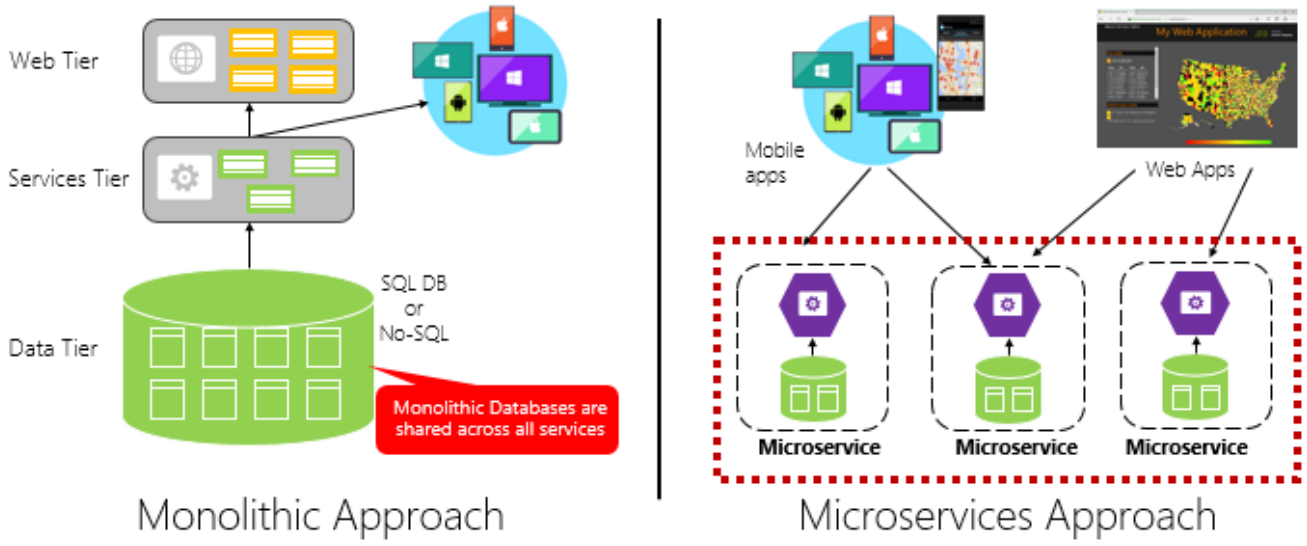


Figura 1-4. Arquitectura monolítica frente a microservicios

Tenga en cuenta cómo los microservicios promueven el principio **Procesos** de la aplicación deTwelve-Factor, descrito anteriormente en el capítulo .

> *Factor 6 especifica "Cada microservicio debe ejecutarse en su propio proceso, aislado de otros servicios en ejecución".*

¿Por qué microservicios?

Los microservicios proporcionan agilidad.

Anteriormente en el capítulo, comparamos una aplicación de comercio electrónico creada como monolítica con los microservicios. En el ejemplo, vimos algunas ventajas claras:

- Cada microservicio tiene un ciclo de vida autónomo y puede evolucionar de forma independiente e implementar con frecuencia. No tiene que esperar a que una versión trimestral implemente una nueva característica o actualización. Puede actualizar un área pequeña de una aplicación activa con menos riesgo de interrumpir todo el sistema. La actualización se puede realizar sin una reimplementación completa de la aplicación.

- Cada microservicio se puede escalar de forma independiente. En lugar de escalar toda la aplicación como una sola unidad, solo se escalan horizontalmente los servicios que requieren más potencia de procesamiento para satisfacer los niveles de rendimiento deseados y los acuerdos de nivel de servicio. El escalado específico proporciona un mayor control del sistema y ayuda a reducir los costos generales a medida que se escalan partes del sistema, no todo.

Una excelente guía de referencia para comprender los microservicios es microservicios de .NET: Arquitectura para aplicaciones .NET en contenedor. El libro profundiza en el diseño y la

arquitectura de microservicios. Es un complemento para una arquitectura de referencia completa de microservicios disponible como descarga gratuita de Microsoft.

Desarrollo de microservicios

Los microservicios se pueden crear en cualquier plataforma de desarrollo moderna.

La plataforma Microsoft .NET es una excelente opción. De código abierto y gratuito, tiene muchas características integradas que simplifican el desarrollo de microservicios. .NET es multiplataforma. Las aplicaciones se pueden compilar y ejecutar en Windows, macOS y la mayoría de los tipos de Linux.

.NET es muy eficaz y ha puntuado bien en comparación con Node.js y otras plataformas de competencia. Interesantemente, TechEmpower llevó a cabo un amplio conjunto de pruebas comparativas de rendimiento en muchas plataformas y marcos de aplicaciones web. .NET se ubicó entre los 10 mejores, muy por encima de Node.js y otras plataformas competidoras.

.NET es mantenido por Microsoft y la comunidad de .NET en GitHub.

Desafíos del microservicio

Aunque los microservicios nativos de nube distribuidos pueden proporcionar una gran agilidad y velocidad, presentan muchos desafíos:

Comunicación

¿Cómo se comunicarán las aplicaciones cliente front-end con microservicios principales de back-end? ¿Permitirá la comunicación directa? O bien, ¿podría abstraer los microservicios de back-end con una fachada de puerta de enlace que proporcione flexibilidad, control y seguridad?

¿Cómo se comunicarán los microservicios principales de back-end entre sí? ¿Permitirá llamadas HTTP directas que puedan aumentar el acoplamiento e impactar el rendimiento y la agilidad? ¿O podría considerar la posibilidad de usar una mensajería desacoplada con tecnologías de cola y tema?

La comunicación se trata en el capítulo Patrones de comunicación nativos de la nube.

Resistencia

Una arquitectura de microservicios mueve su sistema de la comunicación de red dentro del proceso a fuera del proceso. En una arquitectura distribuida, ¿qué ocurre cuando el servicio B

no responde a una llamada de red desde el servicio A? O bien, ¿qué ocurre cuando el servicio C deja de estar disponible temporalmente y otros servicios que lo llaman se bloquean?

La resiliencia se trata en el capítulo Resiliencia nativa en la nube.

Datos distribuidos

Por diseño, cada microservicio encapsula sus propios datos, exponiendo operaciones a través de su interfaz pública. Si es así, ¿cómo se consultan los datos o se implementa una transacción en varios servicios?

Los datos distribuidos se tratan en el capítulo Patrones de datos nativos de la nube.

Secretos

¿Cómo almacenarán y administrarán de forma segura los microservicios y los datos de configuración confidenciales?

Los secretos se tratan en detalle en Seguridad nativa de la nube.

Administración de la complejidad con Dapr

Dapr es un entorno de ejecución de aplicación de código abierto distribuido. Gracias a una arquitectura de componentes que se pueden conectar, simplifica drásticamente los *entresijos* que hay tras las aplicaciones distribuidas. Proporciona un **pegamento dinámico** que enlaza la aplicación con funcionalidades y componentes de infraestructura pregenerados desde el entorno de ejecución de Dapr. En la figura 1-5 se muestra Dapr a 20 000 pies.

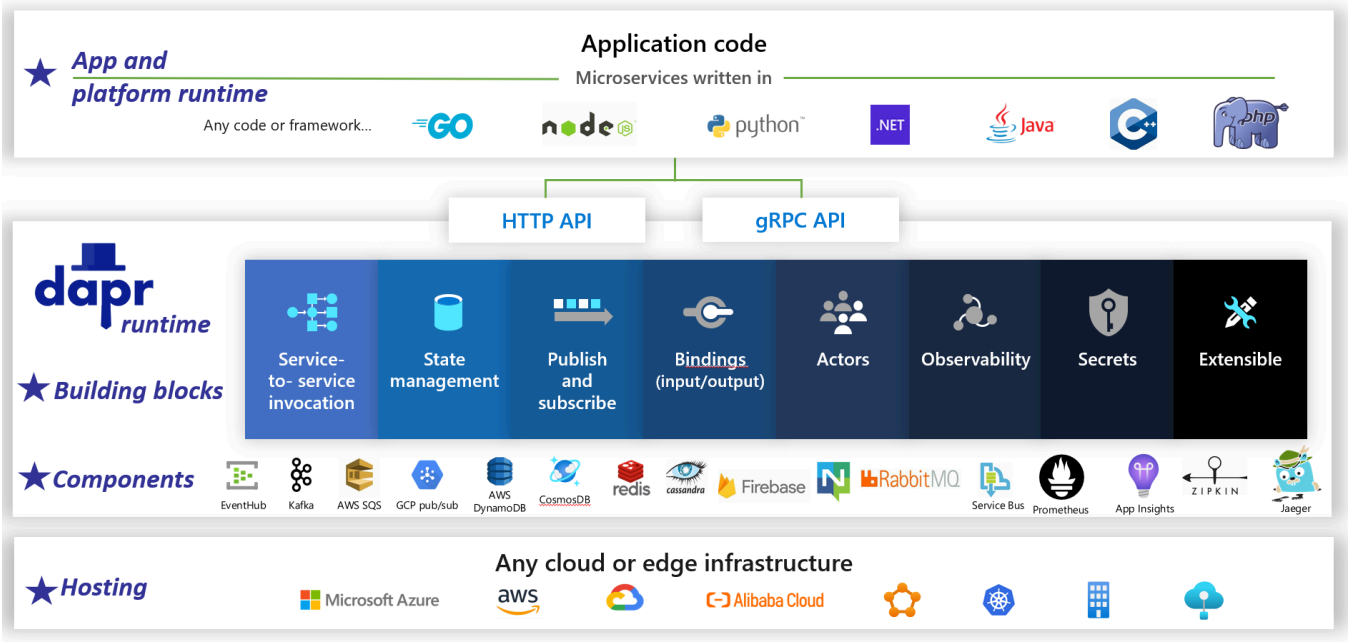


Figura 1-5. Dapr a 20 000 pies.

Fíjese que en la primera fila de la imagen, Dapr proporciona los SDK específicos del lenguaje de varias plataformas de desarrollo conocidas. Dapr v1 incluye compatibilidad con .NET, Go, Node.js, Python, PHP, Java y JavaScript.

Aunque los SDK específicos del lenguaje mejoran la experiencia del desarrollador, Dapr es independiente de la plataforma. En segundo plano, el modelo de programación de Dapr expone funcionalidades a través de protocolos de comunicación HTTP/gRPC estándar. Cualquier plataforma de programación puede llamar a Dapr a través de sus API NATIVAs HTTP y gRPC.

Los cuadros azules en el centro de la imagen son los bloques de creación de Dapr. Cada uno de ellos expone código estructural pregenerado para una funcionalidad de aplicación distribuida que la aplicación puede consumir.

La fila de componentes representa un gran conjunto de componentes de infraestructura predefinidos que la aplicación puede consumir. Piense en los componentes como código de infraestructura que no tiene que escribir.

La fila inferior resalta la portabilidad de Dapr y los diversos entornos en los que se puede ejecutar.

Al mirar adelante, Dapr tiene el potencial de tener un impacto profundo en el desarrollo de aplicaciones nativas de la nube.

Contenedores

Es natural que se mencione el término *contenedor* en cualquier conversación nativa de la *nube*. En el libro, Cloud Native Patterns, el autor Cornelia Davis observa que"Los contenedores son un gran habilitador del software nativo de la nube". Cloud Native Computing Foundation coloca la contenedorización de microservicios como primer paso en su mapa de pistas deCloud-Native : guía para las empresas que comienzan su recorrido nativo de la nube.

La empaquetación de un microservicio en contenedores es fácil y directa. El código, sus dependencias y el entorno de ejecución se empaquetan en un binario denominado imagen de contenedor. Las imágenes se almacenan en un registro de contenedor, que actúa como repositorio o biblioteca para imágenes. Un registro se puede ubicar en el equipo de desarrollo, en el centro de datos o en una nube pública. Docker mantiene un registro público a través de Docker Hub. La nube de Azure incluye un registro de contenedor privado para almacenar imágenes de contenedor cerca de las aplicaciones en la nube que las ejecutarán.

Cuando una aplicación se inicia o escala, transforma la imagen de contenedor en una instancia de contenedor en ejecución. La instancia se ejecuta en cualquier equipo que tenga instalado

un motor en tiempo de ejecución de contenedor. Puede haber tantas instancias del servicio en contenedor como sean necesarias.

En la figura 1-6 se muestran tres microservicios diferentes, cada uno de ellos en su propio contenedor, que se ejecuta en un único host.

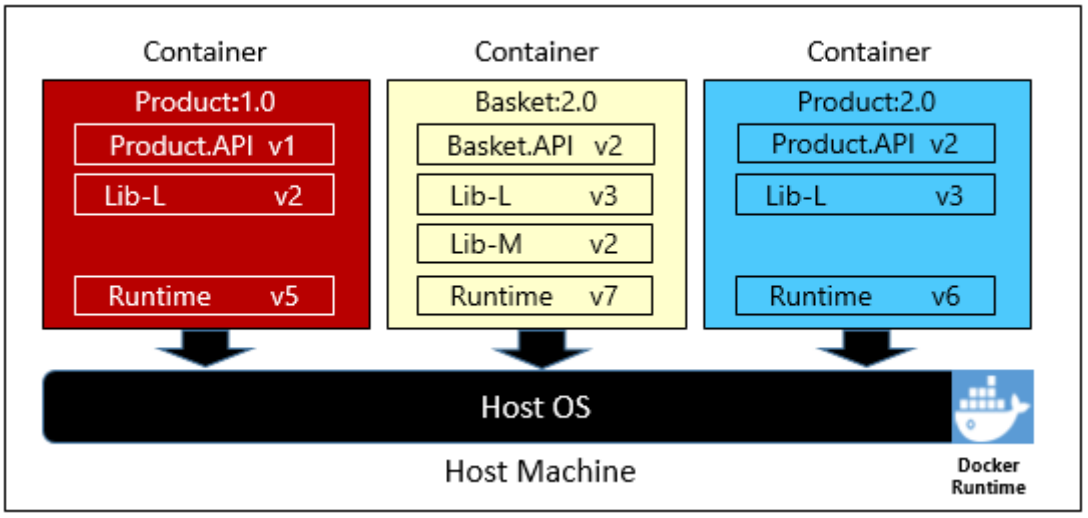


Figura 1-6. Varios contenedores que se ejecutan en un host de contenedor

Tenga en cuenta cómo cada contenedor mantiene su propio conjunto de dependencias y tiempo de ejecución, que puede ser diferente entre sí. Aquí, vemos diferentes versiones del microservicio Product que se ejecuta en el mismo host. Cada contenedor comparte un segmento del sistema operativo del host subyacente, la memoria y el procesador, pero está aislado entre sí.

Tenga en cuenta cómo el modelo de contenedor incorpora el principio de **Dependencias** de la ApplicationTwelve-Factor.

> *Factor 2 especifica que "Cada microservicio aísla y empaqueta sus propias dependencias, adoptando cambios sin afectar a todo el sistema".*

Los contenedores admiten cargas de trabajo de Linux y Windows. La nube de Azure adopta abiertamente ambos. Interesantemente, es Linux, no Windows Server, que se ha convertido en el sistema operativo más popular en Azure.

Mientras existen varios proveedores de contenedores, Docker ha capturado la mayor parte en el mercado. La empresa ha sido una gran impulsora del movimiento de contenedores de software. Se ha convertido en el estándar de facto para empaquetar, implementar y ejecutar aplicaciones nativas de nube.

¿Por qué los contenedores?

Los contenedores proporcionan portabilidad y garantía de coherencia entre entornos. Al encapsular todo en un único paquete, *aísla* el microservicio y sus dependencias de la infraestructura subyacente.

Puede implementar el contenedor en cualquier entorno que hospede el motor en tiempo de ejecución de Docker. Las cargas de trabajo en contenedores también eliminan el gasto de configurar previamente cada entorno con marcos, bibliotecas de software y motores en tiempo de ejecución.

Al compartir los recursos del sistema operativo y del host subyacentes, un contenedor tiene una superficie mucho menor que una máquina virtual completa. El tamaño menor aumenta la *densidad* o el número de microservicios que un host determinado puede ejecutar al mismo tiempo.

Orquestación de contenedores

Aunque las herramientas como Docker crean imágenes y ejecutan contenedores, también necesita herramientas para administrarlas. La administración de contenedores se realiza con un programa de software especial denominado **orquestador de contenedores**. Cuando se trabaja a escala con muchos contenedores que funcionan de manera independiente, la orquestación es esencial.

En la figura 1-7 se muestran las tareas de administración que automatizan los orquestadores de contenedores.

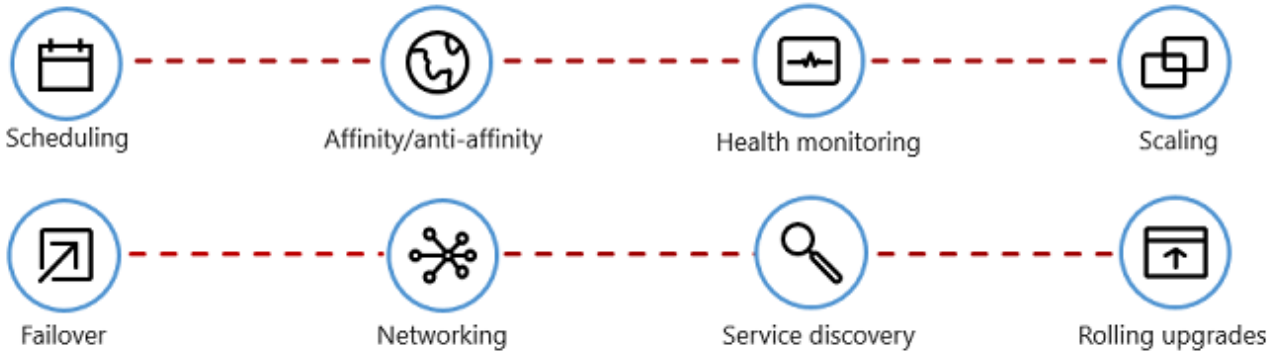


Figura 1-7. Qué hacen los orquestadores de contenedores

En la tabla siguiente se describen las tareas comunes de orquestación.

Expandir tabla

Tareas	Explicación
Programación	Aprovisionamiento automático de instancias de contenedor.

Tareas	Explicación
Afinidad/antiafinidad	Aprovisione contenedores cercanos o alejados entre sí, lo que ayuda a la disponibilidad y el rendimiento.
Monitoreo de la salud	Detectar y corregir errores automáticamente.
Conmutación por error	Aprovisionamiento automático de cualquier instancia con errores en una máquina en buen estado.
Ampliación	Agregue o quite automáticamente una instancia de contenedor para satisfacer la demanda.
Redes	Administre una superposición de red para la comunicación de los contenedores.
Detección de servicios	Habilite los contenedores para localizarse entre sí.
Actualizaciones graduales	Coordinar las actualizaciones incrementales con una implementación sin tiempo de inactividad. Revierta automáticamente los cambios problemáticos.

Tenga en cuenta que los orquestadores de contenedores usan los principios de **Descartabilidad** y **Simultaneidad** de Twelve-Factor Application.

> *Factor #9 especifica que "Las instancias de servicio deben ser descartables, favoreciendo inicios rápidos para aumentar las oportunidades de escalabilidad y apagados elegantes para dejar el sistema en un estado adecuado*". Los contenedores de Docker junto con un orquestador cumplen de manera inherente este requisito".

> *Factor 8 especifica que "Los servicios se escalan horizontalmente en un gran número de pequeños procesos idénticos (copias), en oposición al escalado vertical de una sola instancia grande en la máquina disponible de mayor potencia*".

Aunque existen varios orquestadores de contenedores, Kubernetes se ha convertido en el estándar de facto para el mundo nativo de la nube. Es una plataforma portable, extensible y de código abierto para administrar cargas de trabajo en contenedores.

Podría hospedar su propia instancia de Kubernetes, pero después sería responsable del aprovisionamiento y la administración de sus recursos, lo que puede ser complejo. La nube de Azure incluye Kubernetes como servicio administrado. Tanto Azure Kubernetes Service (AKS) como Red Hat OpenShift (ARO) de Azure le permiten aprovechar completamente las características y la eficacia de Kubernetes como servicio administrado, sin tener que instalarla y mantenerla.

La orquestación de contenedores se trata en detalle en Escalado de aplicaciones nativas de la nube.

Servicios de respaldo

Los sistemas nativos de nube dependen de muchos recursos auxiliares diferentes, como almacenes de datos, agentes de mensajes, supervisión y servicios de identidad. Estos servicios se conocen como servicios de respaldo.

En la figura 1-8 se muestran muchos servicios de respaldo comunes que consumen los sistemas nativos de la nube.

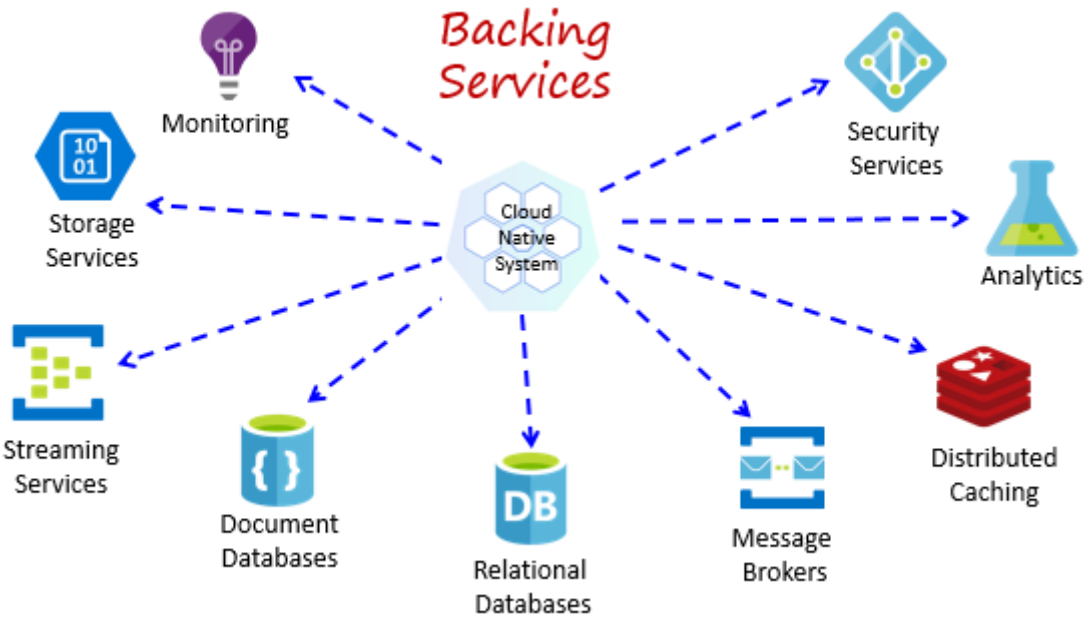


Figura 1-8. Servicios de respaldo comunes

Puede hospedar sus propios servicios de respaldo, pero después sería responsable de las licencias, el aprovisionamiento y la administración de esos recursos.

Los proveedores de nube ofrecen una amplia variedad de *servicios de respaldo administrados*. En lugar de poseer el servicio, simplemente lo consume. El proveedor de nube opera el recurso a escala y asume la responsabilidad del rendimiento, la seguridad y el mantenimiento. La supervisión, la redundancia y la disponibilidad están integradas en el servicio. Los proveedores garantizan el rendimiento del nivel de servicio y respaldan completamente sus servicios gestionados: abra una incidencia y solucionan su problema.

Los sistemas nativos en la nube favorecen los servicios de respaldo administrados de los proveedores de nube. El ahorro en el tiempo y la mano de obra pueden ser significativos. El riesgo operativo de gestionar tu propio servidor y si surgen problemas, puede volverse rápidamente costoso.

Un procedimiento recomendado es tratar un servicio de respaldo como un **recurso asociado**, enlazado dinámicamente a un microservicio con información de configuración (una dirección

URL y credenciales) almacenada en una configuración externa. Esta guía se explica en la Twelve-Factor Aplicación, que se ha descrito anteriormente en el capítulo .

> *Factor 4* especifica que los servicios de respaldo "deben exponerse a través de una dirección URL direccionable. Si lo hace, desacopla el recurso de la aplicación, lo que le permite ser intercambiable".

> *Factor #3* especifica que "La información de configuración se mueve fuera del microservicio y se externaliza a través de una herramienta de administración de configuración fuera del código".

Con este patrón, un servicio de respaldo se puede adjuntar y desasociar sin cambios de código. Cualquier microservicio se puede promover de QA a un entorno de ensayo. Actualice la configuración del microservicio para que apunte a los servicios de respaldo del almacenamiento provisional e inserte la configuración en el contenedor a través de una variable de entorno.

Los proveedores en la nube proporcionan API para comunicarse con sus servicios de respaldo propietarios. Estas bibliotecas encapsulan la complejidad y los entresijos de los servicios propietarios. Sin embargo, la comunicación directa con estas API acoplará estrechamente el código a ese servicio de respaldo específico. Es una práctica ampliamente aceptada para aislar los detalles de implementación de la API del proveedor. Introduzca una capa de intermediación o una API intermedia, exponiendo operaciones genéricas al código de servicio y encapsulando el código del proveedor dentro de él. Este acoplamiento flexible le permite intercambiar un servicio de respaldo para otro o mover el código a otro entorno en la nube sin tener que realizar cambios en el código de servicio de línea principal. Dapr, descrito anteriormente, sigue este modelo con su conjunto de bloques de creación precompilados.

Por último, los servicios de respaldo también promueven el principio de **Falta de estado** de Twelve-Factor Application, que ya se ha explicado en este mismo capítulo.

> *Factor #6* especifica que" Cada microservicio debe ejecutarse en su propio proceso, aislado de otros servicios en ejecución. Externalice el estado necesario a un servicio de respaldo, como una caché distribuida o un almacén de datos".

Los servicios de respaldo se describen en patrones de datos nativos de la nube y patrones de comunicación nativos de la nube.

Automatización

Como ha visto, los sistemas nativos de la nube adoptan microservicios, contenedores y diseño moderno del sistema para lograr velocidad y agilidad. Pero eso es solo parte de la historia. ¿Cómo aprovisiona los entornos en la nube en los que se ejecutan estos sistemas? ¿Cómo se implementan rápidamente las características y actualizaciones de la aplicación? ¿Cómo se completa la imagen?

Introducción a la práctica ampliamente aceptada de Infraestructura como código, o IaC.

Con IaC, automatiza el aprovisionamiento de plataformas y la implementación de aplicaciones. Básicamente, aplica prácticas de ingeniería de software, como pruebas y control de versiones a las prácticas de DevOps. La infraestructura y las implementaciones son automatizadas, coherentes y repetibles.

Automatización de la infraestructura

Herramientas como Azure Resource Manager, Azure Bicep, Terraform desde HashiCorp y la CLI de Azure, le permiten crear scripts declarativos de la infraestructura en la nube que necesita. Los nombres de recursos, las ubicaciones, las capacidades y los secretos son parametrizados y dinámicos. El script tiene versiones y se inserta en el repositorio del control de código fuente como un artefacto del proyecto. El script se invoca para aprovisionar una infraestructura coherente y repetible en entornos del sistema, como QA, ensayo y producción.

En segundo plano, IaC es idempotente, lo que significa que se puede ejecutar el mismo script una y otra vez sin efectos secundarios. Si el equipo necesita realizar un cambio, edita y vuelve a ejecutar el script. Solo se ven afectados los recursos actualizados.

En el artículo ¿Qué es la infraestructura como código?, el autor sam Guckenheimer describe cómo" Los equipos que implementan IaC pueden ofrecer entornos estables rápidamente y a escala. Evitan la configuración manual de entornos y aplican la coherencia mediante la representación del estado deseado de sus entornos a través del código. Las implementaciones de infraestructura con IaC son repetibles y evitan problemas en tiempo de ejecución causados por el desfase de configuración o las dependencias que faltan. Los equipos de DevOps pueden trabajar conjuntamente con un conjunto unificado de prácticas y herramientas para ofrecer aplicaciones y su infraestructura auxiliar de forma rápida, confiable y a gran escala".

Automatización de implementaciones

La Twelve-Factor Aplicación, que se ha descrito anteriormente, llama a pasos independientes al transformar el código completado en una aplicación en ejecución.

> *Factor 5* especifica que "Cada versión debe aplicar una separación estricta entre las fases de compilación, versión y ejecución. Cada una de ellas debe etiquetarse con un

> identificador único y permitir la reversión".

Los sistemas de CI/CD modernos ayudan a cumplir este principio. Proporcionan pasos de compilación y entrega independientes que ayudan a garantizar un código coherente y de calidad que esté disponible fácilmente para los usuarios.

En la figura 1-9 se muestra la separación a lo largo del proceso de implementación.

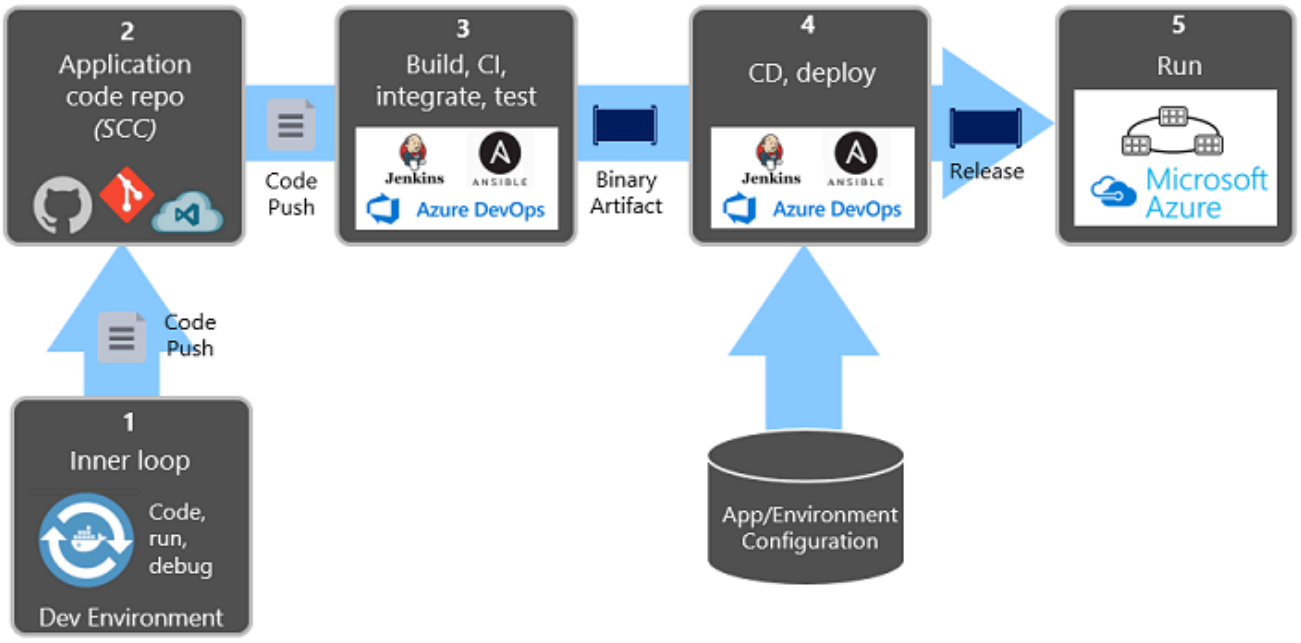


Figura 1-9. Pasos de implementaciones en una canalización de integración continua y entrega continua

En la ilustración anterior, preste especial atención a la separación de tareas:

1. El desarrollador crea una característica en su entorno de desarrollo, iterando a través de lo que se denomina "bucle interno" de código, ejecución y depuración.
2. Cuando haya finalizado, ese código se *inserta* en un repositorio de código, como GitHub, Azure DevOps o BitBucket.
3. El envío desencadena una fase de compilación que transforma el código en un artefacto binario. El trabajo se implementa con una canalización de integración continua (CI). Compila, prueba y empaqueta automáticamente la aplicación.
4. La fase de versión recoge el artefacto binario, aplica información de configuración de entorno y aplicación externa y genera una versión inmutable. La versión se implementa en un entorno especificado. El trabajo se implementa con una canalización de entrega continua (CD). Cada versión debe poder identificarse fácilmente. Puede decir: "Esta implementación ejecuta la versión 2.1.1 de la aplicación".
5. Por último, la característica publicada se ejecuta en el entorno de ejecución de destino. Las versiones son inmutables, lo que significa que cualquier cambio debe crear una nueva

versión.

Al aplicar estas prácticas, las organizaciones han evolucionado radicalmente cómo envían software. Muchos han pasado de las versiones trimestrales a las actualizaciones a petición. El objetivo es detectar problemas al principio del ciclo de desarrollo cuando son menos costosos de corregir. Cuanto mayor sea la duración entre las integraciones, más costoso se vuelve resolver los problemas. Con la coherencia en el proceso de integración, los equipos pueden confirmar los cambios de código con más frecuencia, lo que conduce a una mejor colaboración y calidad de software.

La infraestructura como automatización de código e implementación, junto con GitHub y Azure DevOps, se describen en detalle en DevOps.

Siguiente anterior

Aplicaciones candidatas para la nube nativa

17/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Piense en las aplicaciones que su organización necesita desarrollar. A continuación, examine las aplicaciones existentes en la cartera. ¿Cuántos de ellos garantizan una arquitectura nativa de la nube? ¿Todos ellos? ¿Quizás algunos?

Al aplicar el análisis de costes y beneficios, podría ser que algunos no apoyen el esfuerzo. El costo de convertirse en nativo de nube superaría mucho el valor empresarial de la aplicación.

¿Qué tipo de aplicación puede ser candidata para la nube nativa?

- Sistemas empresariales estratégicos que necesitan evolucionar constantemente las funcionalidades y características empresariales
- Una aplicación que requiere una alta velocidad de implementación, con alta confianza
- Un sistema en el que las características individuales deben liberarse *sin* una reimplementación completa del sistema completo
- Una aplicación desarrollada por equipos con experiencia en diferentes entornos tecnológicos
- Una aplicación con componentes que deben escalarse de forma independiente

Las aplicaciones de línea de negocio más pequeñas y menos impactantes pueden resultar muy bien con una arquitectura monolítica simple hospedada en un entorno paaS en la nube.

A continuación, hay sistemas heredados. Aunque nos gustaría desarrollar nuevas aplicaciones, a menudo somos responsables de modernizar las cargas de trabajo heredadas críticas para el negocio.

Modernización de aplicaciones heredadas

El libro electrónico gratuito de Microsoft Modernize existing .NET applications with Azure cloud and Windows Containers (Modernize existing .NET applications with Azure cloud and Windows Containers) proporciona instrucciones sobre cómo migrar cargas de trabajo locales a la nube. En la figura 1-10 se muestra que no hay una estrategia única y única para modernizar las aplicaciones heredadas.

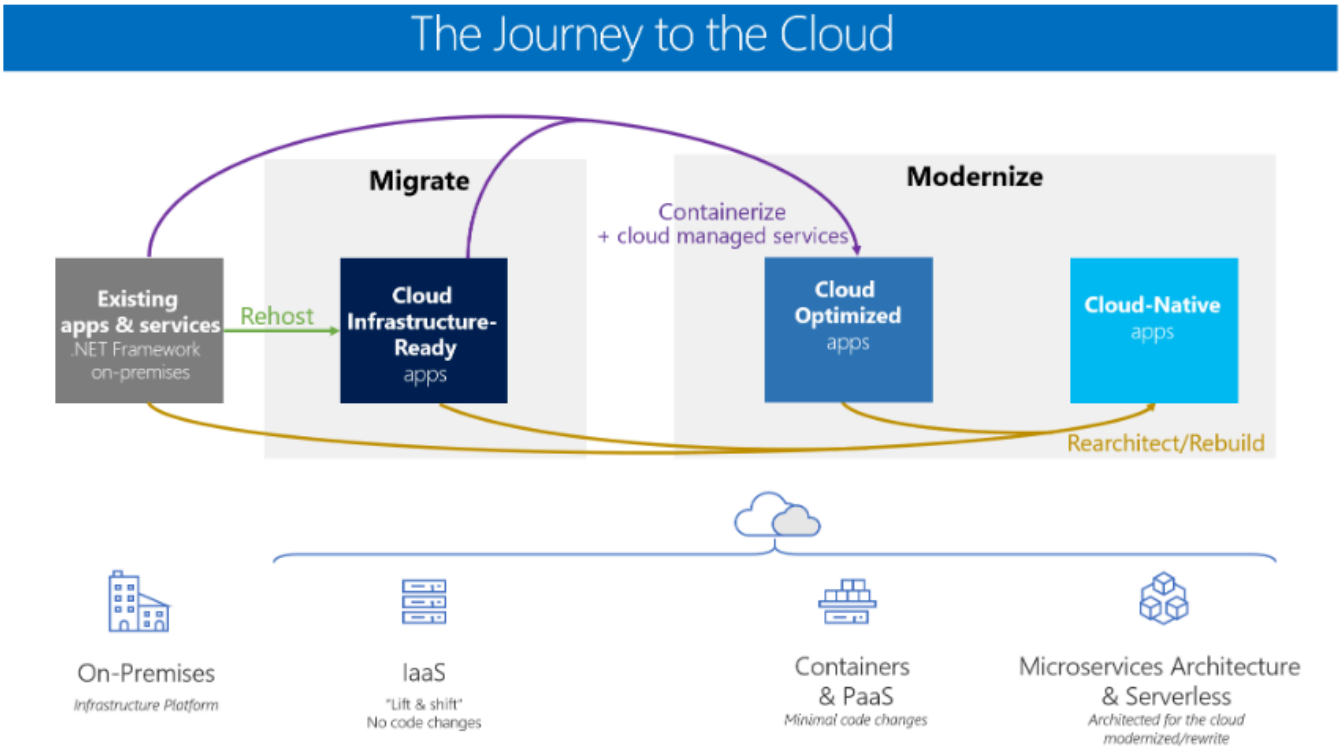


Figura 1-10. Estrategias para migrar cargas de trabajo heredadas

Es posible que las aplicaciones monolíticas que no son críticas se beneficien de una migración rápida **mediante lift-and-shift**. Aquí, la carga de trabajo local se rehospeda en una máquina virtual basada en la nube, sin cambios. Este enfoque usa el modelo iaaS (infraestructura como servicio). Azure incluye varias herramientas como Azure Migrate, Azure Site Recovery y Azure Database Migration Service para ayudar a simplificar el traslado. Aunque esta estrategia puede producir algunos ahorros de costos, estas aplicaciones normalmente no estaban diseñadas para desbloquear y aprovechar las ventajas de la informática en la nube.

Las aplicaciones heredadas que son críticas para la empresa suelen beneficiarse de una migración mejorada **optimizada para la nube** . Este enfoque incluye optimizaciones de implementación que habilitan los servicios en la nube clave, sin cambiar la arquitectura principal de la aplicación. Por ejemplo, puede incluir en contenedores la aplicación e

implementarla en un orquestador de contenedores, como Azure Kubernetes Services, que se describe más adelante en este libro. Una vez en la nube, la aplicación puede consumir servicios de respaldo en la nube, como bases de datos, colas de mensajes, supervisión y almacenamiento en caché distribuido.

Por último, las aplicaciones monolíticas que proporcionan funciones empresariales estratégicas pueden beneficiarse mejor de un enfoque nativo de la *nube* , el tema de este libro. Este enfoque proporciona agilidad y velocidad. A cambio, conlleva un cambio de plataforma, el rediseño de la arquitectura y la reescritura de código. Con el tiempo, una aplicación heredada se podría descomponer en microservicios, en contenedores y, en última instancia, *volver a plataformarse* en una arquitectura nativa de la nube.

Si usted y su equipo creen que un enfoque nativo de la nube es adecuado, le corresponde racionalizar la decisión con su organización. ¿Cuál es exactamente el problema empresarial que resolverá un enfoque nativo de la nube? ¿Cómo se alinearía con las necesidades empresariales?

- ¿Lanzamientos rápidos de funcionalidades con mayor confianza?
- Escalabilidad específica: ¿un uso más eficaz de los recursos?
- ¿Se ha mejorado la resistencia del sistema?
- ¿Se ha mejorado el rendimiento del sistema?
- ¿Más visibilidad de las operaciones?
- ¿Combina plataformas de desarrollo y almacenes de datos para llegar a la mejor herramienta para el trabajo?
- ¿Inversión en aplicaciones a prueba de futuro?

La estrategia de migración adecuada depende de las prioridades organizativas y de los sistemas a los que se dirige. Para muchos, puede ser más rentable optimizar en la nube una aplicación monolítica o agregar servicios generales a una aplicación de N niveles. En estos casos, todavía puede usar todas las funcionalidades de PaaS en la nube, como las que ofrece Azure App Service.

Resumen

En este capítulo, presentamos la informática nativa de la nube. Se ha proporcionado una definición junto con las funcionalidades clave que impulsan una aplicación nativa de la nube. Hemos examinado los tipos de aplicaciones que podrían justificar esta inversión y esfuerzo.

Tras esta introducción, profundizaremos en detalle en el concepto "nativo de nube".

Referencias

- Cloud Native Computing Foundation
- Microservicios de .NET: arquitectura para aplicaciones .NET en contenedor
- Microsoft Azure Well-Architected Framework
- Modernización de aplicaciones .NET existentes con contenedores en la nube de Azure y Windows
- Patrones nativos en la nube de Cornelia Davis
- Aplicaciones nativas en la nube: envío más rápido, reducción del riesgo y crecimiento de su negocio
- Documentos Dapr
- Más allá de la metodología Twelve-Factor
- ¿Qué es la infraestructura como código?
- Micro Deploy de Uber Engineering: Despliegue diario con confianza
- Cómo implementa Netflix Código
- Control de sobrecarga para el escalado de microservicios de WeChat

Siguiente anterior

Presentación de la aplicación de referencia de eShopOnContainers

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Microsoft, en asociación con expertos líderes de la comunidad, ha producido una aplicación de referencia completa de microservicios nativos en la nube, eShopOnContainers. Esta aplicación está diseñada para demostrar el uso de .NET y Docker, y opcionalmente Azure, Kubernetes y Visual Studio, para desarrollar una tienda en línea.

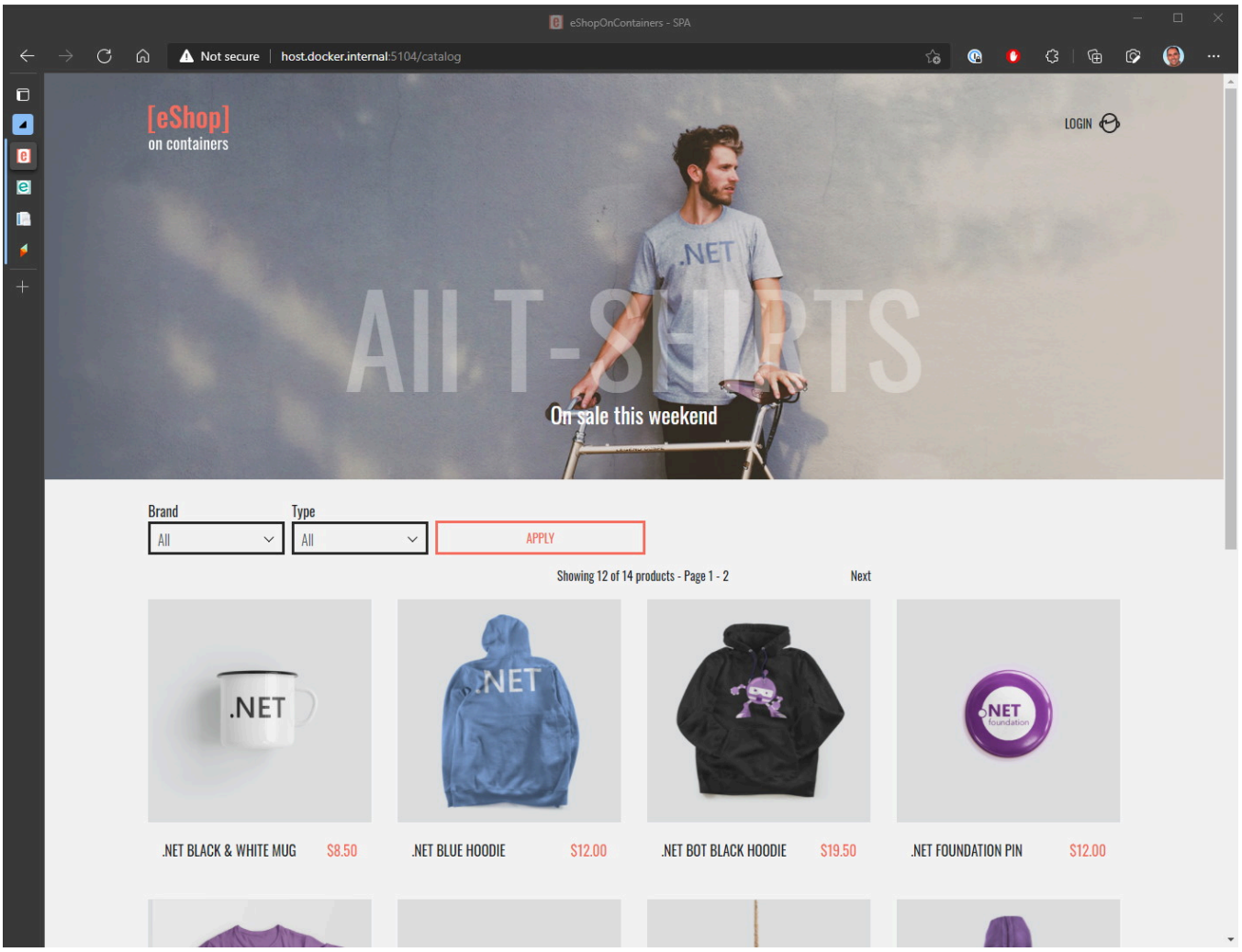


Figura 2-1. Captura de pantalla de la aplicación de ejemplo de eShopOnContainers.

Antes de iniciar este capítulo, se recomienda descargar la aplicación de referencia eShopOnContainers. Si lo hace, debería ser más fácil seguir la información presentada.

Características y requisitos

Comencemos con una revisión de las características y requisitos de la aplicación. La aplicación eShopOnContainers representa una tienda en línea que vende varios productos físicos como camisetas y tazas de café. Si has comprado algo en línea antes, la experiencia de usar la tienda debe ser relativamente familiar. Estas son algunas de las características básicas que implementa la tienda:

- Enumerar elementos de catálogo
- Filtrar elementos por tipo
- Filtrar elementos por marca
- Agregar artículos a la cesta de la compra
- Editar o quitar elementos de la cesta
- Restauración
- Registro de una cuenta

- Iniciar sesión
- Cerrar sesión
- Revisar pedidos

La aplicación también tiene los siguientes requisitos no funcionales:

- Debe tener alta disponibilidad y escalar automáticamente para satisfacer el aumento del tráfico (y desescalar automáticamente una vez que se reduzca el tráfico).
- Debe proporcionar una supervisión fácil de usar de sus registros de estado y diagnóstico para ayudar a solucionar los problemas que encuentre.
- Debe admitir un proceso de desarrollo ágil, incluida la compatibilidad con la integración e implementación continuas (CI/CD).
- Además de los dos front-end web (aplicación tradicional y de página única), la aplicación también debe admitir aplicaciones cliente móviles que ejecutan diferentes tipos de sistemas operativos.
- Debe admitir el hospedaje multiplataforma y el desarrollo multiplataforma.

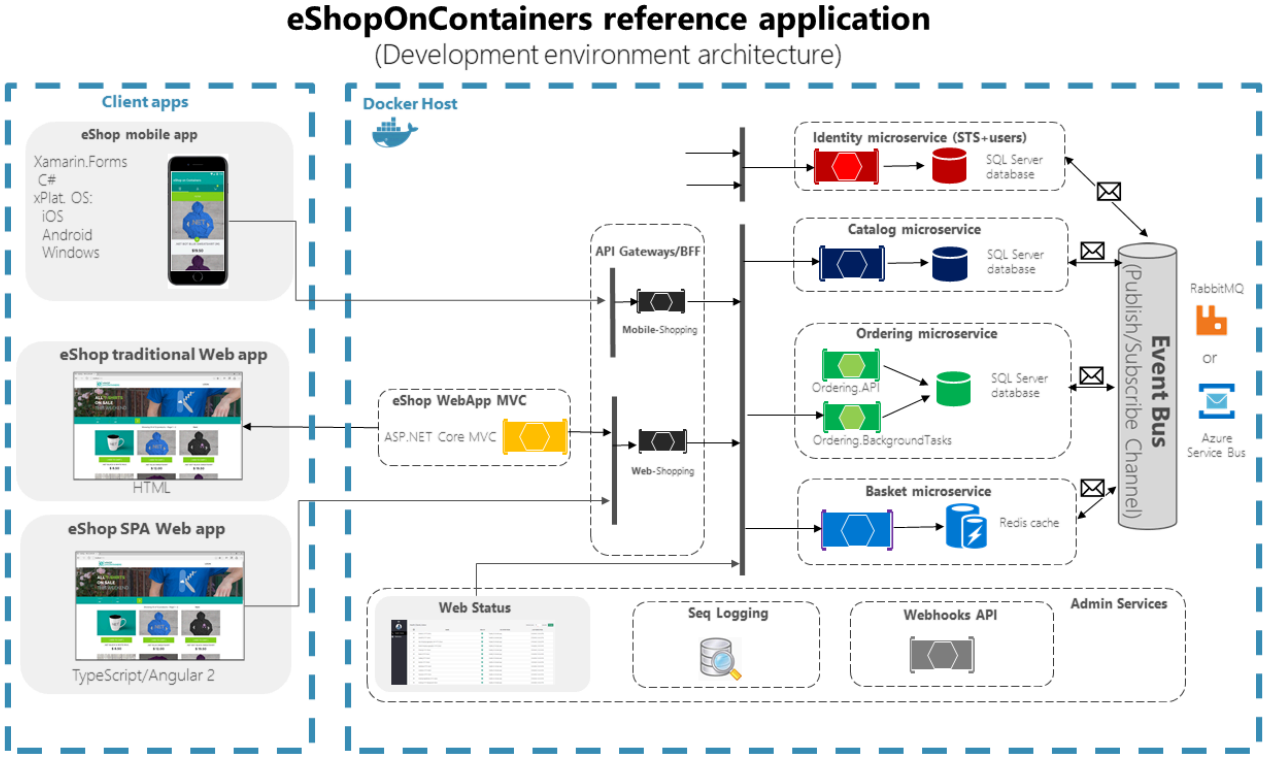


Figura 2-2. La arquitectura de desarrollo de aplicaciones de referencia de eShopOnContainers.

La aplicación eShopOnContainers es accesible desde clientes web o móviles que acceden a la aplicación a través de HTTPS que tienen como destino la aplicación de servidor ASP.NET Core MVC o una puerta de enlace de API adecuada. Las puertas de enlace de API ofrecen varias ventajas, como desacoplar servicios back-end de clientes front-end individuales y proporcionar una mejor seguridad. La aplicación también usa un patrón relacionado conocido como backends-for-Frontends (BFF), que recomienda crear puertas de enlace de API independientes

para cada cliente front-end. La arquitectura de referencia muestra la separación de las puertas de enlace de API en función de si la solicitud procede de un cliente web o móvil.

La funcionalidad de la aplicación se divide en muchos microservicios distintos. Hay servicios responsables de la autenticación y la identidad, enumerar artículos del catálogo de productos, administrar las cestas de compra de los usuarios y realizar pedidos. Cada uno de estos servicios independientes tiene su propio almacenamiento persistente. No hay ningún almacén de datos principal único con el que interactúan todos los servicios. En su lugar, la coordinación y la comunicación entre los servicios se realizan según sea necesario y mediante un bus de mensajes.

Cada uno de los diferentes microservicios está diseñado de forma diferente, en función de sus requisitos individuales. Este aspecto significa que su pila de tecnología puede diferir, aunque todas se hayan creado con .NET y estén diseñadas para la nube. Los servicios más sencillos proporcionan acceso básico Crear-Leer-Actualizar-Eliminar (CRUD) a los almacenes de datos subyacentes, mientras que los servicios más avanzados usan patrones y enfoques de diseño guiado por el dominio para administrar la complejidad de negocio.

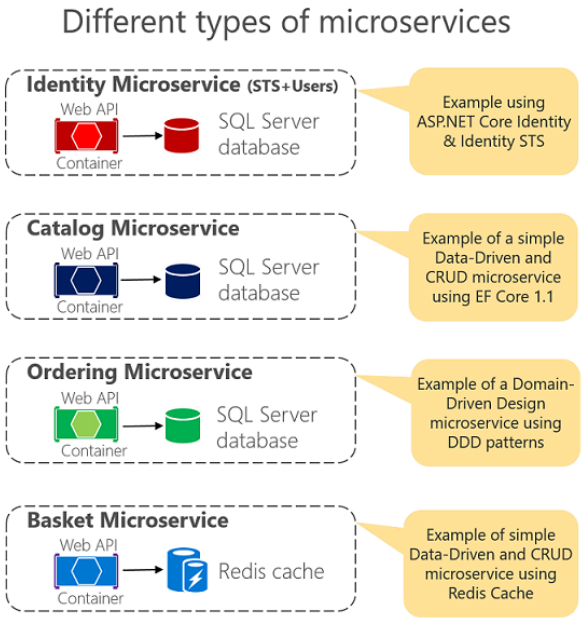


Figura 2-3. Diferentes tipos de microservicios.

Información general sobre el código

Dado que usa microservicios, la aplicación eShopOnContainers incluye bastantes proyectos y soluciones independientes en su repositorio de GitHub. Además de soluciones independientes y archivos ejecutables, los distintos servicios están diseñados para ejecutarse dentro de sus propios contenedores, tanto durante el desarrollo local como en tiempo de ejecución en producción. En la figura 2-4 se muestra la solución completa de Visual Studio, en la que se organizan los distintos proyectos.

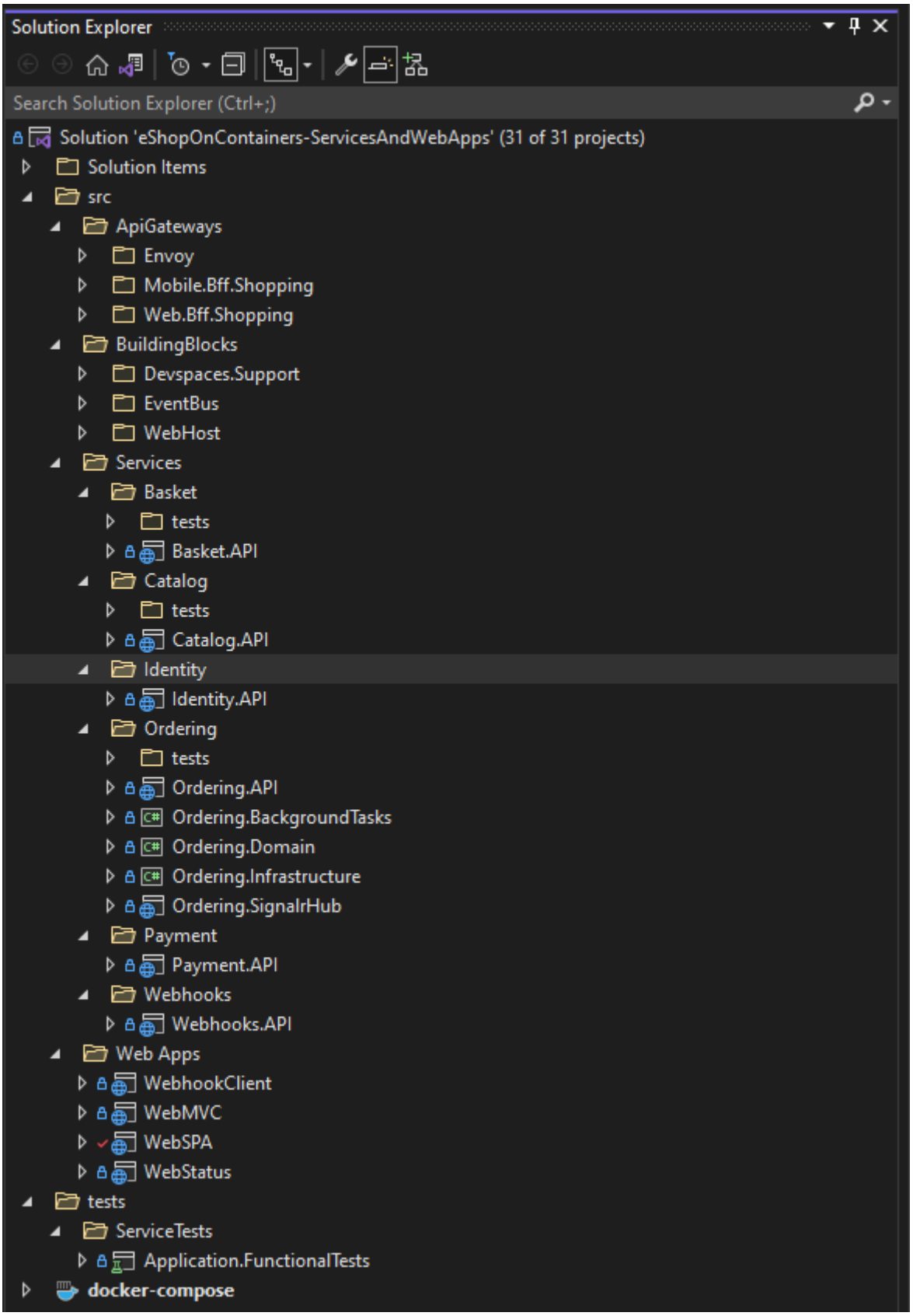


Figura 2-4. Proyectos en la solución de Visual Studio.

El código se organiza para admitir los diferentes microservicios y, dentro de cada microservicio, el código se divide en lógica de dominio, problemas de infraestructura y interfaz de usuario o punto de conexión de servicio. En muchos casos, las dependencias de cada servicio pueden cumplirse mediante servicios de Azure en producción, así como opciones alternativas para el

desarrollo local. Examinemos cómo se asignan los requisitos de la aplicación a los servicios de Azure.

Descripción de los microservicios

Este libro se centra en las aplicaciones nativas de la nube creadas mediante la tecnología de Azure. Para obtener más información sobre los procedimientos recomendados de microservicios y cómo diseñar aplicaciones basadas en microservicios, lea el libro complementario Microservicios de .NET: Arquitectura para aplicaciones .NET en contenedor.

Siguiente anterior

Mapeo de eShopOnContainers con los Servicios de Azure

19/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Aunque no es necesario, Azure es adecuado para admitir eShopOnContainers porque el proyecto se creó para ser una aplicación nativa de la nube. La aplicación se compila con .NET, por lo que puede ejecutarse en contenedores de Linux o Windows en función del host de Docker. La aplicación se compone de varios microservicios autónomos, cada uno con sus propios datos. Los distintos microservicios muestran diferentes enfoques, desde operaciones CRUD simples hasta patrones DDD y CQRS más complejos. Los microservicios se comunican con clientes a través de HTTP y entre sí a través de la comunicación basada en mensajes. La aplicación también admite varias plataformas para clientes, ya que adopta HTTP como protocolo de comunicación estándar e incluye ASP.NET aplicaciones móviles de Core y Xamarin que se ejecutan en plataformas Android, iOS y Windows. (Xamarin deja de ser compatible a partir de mayo de 2024).

La arquitectura de la aplicación se muestra en la figura 2-5. A la izquierda se encuentran las aplicaciones cliente, divididas en tipos de aplicación de página única (SPA) móviles, tradicionales y web. A la derecha se encuentran los componentes del lado servidor que componen el sistema, cada uno de los cuales se puede hospedar en contenedores de Docker y clústeres de Kubernetes. La aplicación web tradicional se basa en la aplicación ASP.NET Core MVC que se muestra en amarillo. Esta aplicación y las aplicaciones spa móviles y web se comunican con los microservicios individuales a través de una o varias puertas de enlace de API. Las puertas de enlace de API siguen el patrón "back-ends para front-ends", lo que significa

que cada puerta de enlace está diseñada para admitir un cliente front-end determinado. Los microservicios individuales aparecen a la derecha de las puertas de enlace de API, e incluyen tanto la lógica de negocios como algún tipo de almacén de persistencia. Los distintos servicios usan bases de datos de SQL Server, instancias de caché de Redis y almacenes de MongoDB/CosmosDB. En el extremo derecho se encuentra el bus de eventos del sistema, que se usa para la comunicación entre los microservicios.

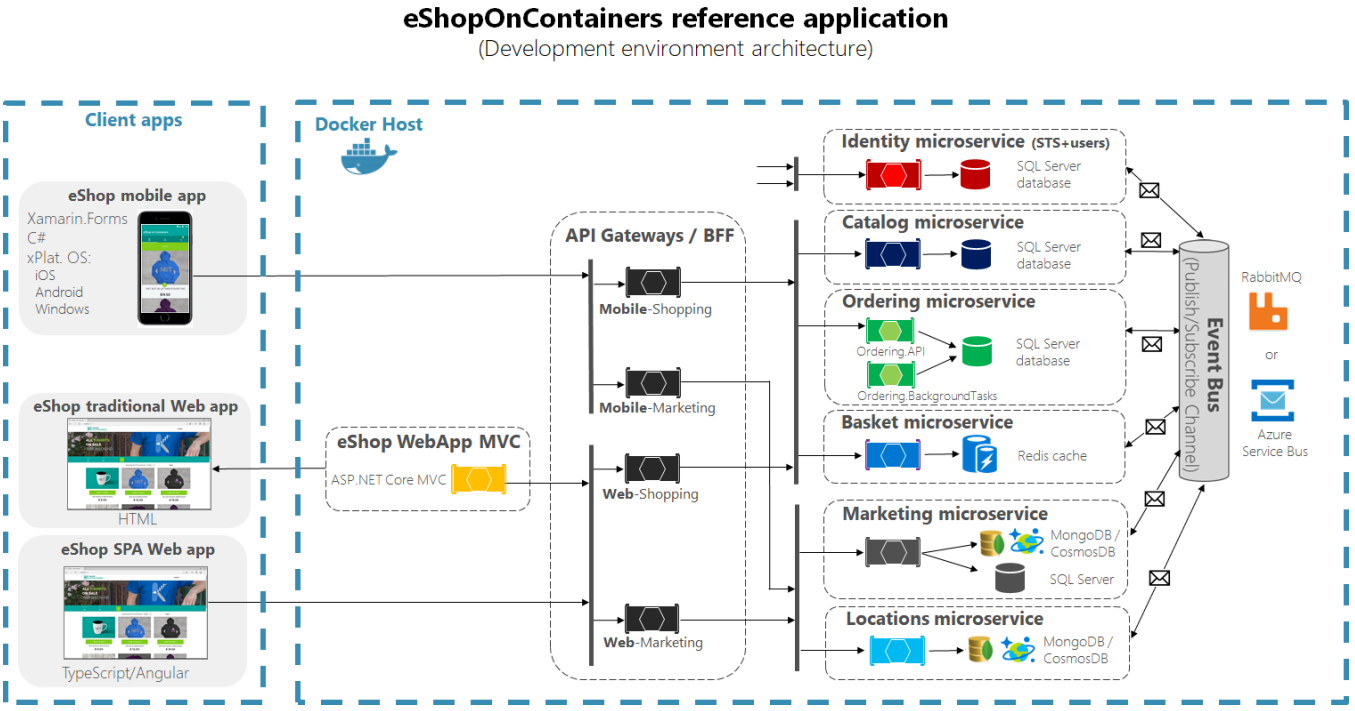


Figura 2-5. La arquitectura de eShopOnContainers.

Los componentes del lado servidor de esta arquitectura se asignan fácilmente a los servicios de Azure.

Orquestación de contenedores y agrupación en clústeres

Los servicios hospedados en contenedores de la aplicación, desde aplicaciones de ASP.NET Core MVC a microservicios individuales de catálogo y pedidos, se pueden hospedar y administrar en Azure Kubernetes Service (AKS). La aplicación se puede ejecutar localmente en Docker y Kubernetes, y los mismos contenedores se pueden implementar en entornos de ensayo y producción hospedados en AKS. Este proceso se puede automatizar, como veremos en la sección siguiente.

AKS proporciona servicios de administración para clústeres individuales de contenedores. La aplicación implementará contenedores independientes para cada microservicio en el clúster de AKS, como se muestra en el diagrama de arquitectura anterior. Este enfoque permite que cada servicio individual se escale de forma independiente según sus demandas de recursos. Cada

microservicio también se puede implementar de forma independiente y, idealmente, estas implementaciones deben incurrir en un tiempo de inactividad del sistema cero.

Puerta de enlace de API

La aplicación eShopOnContainers tiene varios clientes front-end y varios servicios back-end diferentes. No hay correspondencia uno a uno entre las aplicaciones cliente y los microservicios que los admiten. En este escenario, puede haber una gran complejidad al escribir software cliente para interactuar con los distintos servicios back-end de forma segura. Cada cliente tendría que abordar esta complejidad por su cuenta, lo que da lugar a la duplicación y a muchos lugares en los que realizar actualizaciones a medida que se implementan los cambios de servicios o se implementan nuevas directivas.

Azure API Management (APIM) ayuda a las organizaciones a publicar API de forma coherente y manejable. APIM consta de tres componentes: la puerta de enlace de API y el portal de administración (Azure Portal) y un portal para desarrolladores.

API Gateway acepta llamadas API y las enruta a la API de back-end adecuada. También puede proporcionar servicios adicionales, como la comprobación de claves de API o tokens JWT y transformación de API sobre la marcha sin modificaciones de código (por ejemplo, para dar cabida a los clientes que esperan una interfaz anterior).

Azure Portal es donde se definen el esquema de API y se empaquetan diferentes API en productos. También puede configurar el acceso de usuario, ver informes y configurar directivas para cuotas o transformaciones.

El portal para desarrolladores actúa como el recurso principal para los desarrolladores. Proporciona a los desarrolladores documentación de API, una consola de prueba interactiva e informes sobre su propio uso. Los desarrolladores también usan el portal para crear y administrar sus propias cuentas, incluida la compatibilidad con la suscripción y la clave de API.

Con APIM, las aplicaciones pueden exponer varios grupos de servicios diferentes, cada uno de los cuales proporciona un back-end para un cliente front-end determinado. SE recomienda APIM para escenarios complejos. Para las necesidades más sencillas, se puede usar la puerta de enlace de API ligera Ocelot. La aplicación eShopOnContainers usa Ocelot debido a su simplicidad y porque se puede implementar en el mismo entorno de aplicación que la propia aplicación. Obtenga más información sobre eShopOnContainers, APIM y Ocelot.

Otra opción, si tu aplicación está usando AKS, es implementar Azure Gateway Ingress Controller como un pod dentro del clúster de AKS. Este enfoque permite que el clúster se integre con una Azure Application Gateway, lo que permite que la puerta de enlace equilibre la

carga del tráfico a los pods de AKS. Obtenga más información sobre el controlador de entrada de puerta de enlace de Azure para AKS.

Datos

Los distintos servicios back-end usados por eShopOnContainers tienen requisitos de almacenamiento diferentes. Varios microservicios usan bases de datos de SQL Server. El microservicio Basket aprovecha una caché de Redis para su persistencia. El microservicio Locations espera una API de MongoDB para sus datos. Azure admite cada uno de estos formatos de datos.

Para la compatibilidad con bases de datos de SQL Server, Azure tiene productos para todo, desde bases de datos únicas hasta grupos elásticos de SQL Database altamente escalables. Los microservicios individuales se pueden configurar para comunicarse con sus propias bases de datos individuales de SQL Server de forma rápida y sencilla. Estas bases de datos se pueden escalar según sea necesario para admitir cada microservicio independiente según sus necesidades.

La aplicación eShopOnContainers almacena la cesta de compras actual del usuario entre solicitudes. Este aspecto se administra mediante el microservicio Basket que almacena los datos en una caché de Redis. En desarrollo, esta caché se puede implementar en un contenedor, mientras que en producción puede usar Azure Cache for Redis. Azure Cache for Redis es un servicio totalmente administrado que ofrece un alto rendimiento y confiabilidad sin necesidad de implementar y administrar instancias o contenedores de Redis por su cuenta.

El microservicio Locations usa una base de datos NoSQL de MongoDB para su persistencia. Durante el desarrollo, la base de datos se puede implementar en su propio contenedor, mientras que en producción, el servicio puede aprovechar la API de Azure Cosmos DB para MongoDB. Una de las ventajas de Azure Cosmos DB es su capacidad de aprovechar varios protocolos de comunicación diferentes, como una API de SQL y api noSQL comunes, como MongoDB, Cassandra, Gremlin y Azure Table Storage. Azure Cosmos DB ofrece una base de datos como servicio totalmente administrada y distribuida globalmente que puede escalar para satisfacer las necesidades de los servicios que lo usan.

Los datos distribuidos en aplicaciones nativas de nube se tratan con más detalle en el capítulo 5.

Bus de eventos

La aplicación usa eventos para comunicar los cambios entre distintos servicios. Esta funcionalidad se puede implementar con varias implementaciones y la aplicación

eShopOnContainers usa RabbitMQ localmente. Cuando se hospeda en Azure, la aplicación aprovecharía Azure Service Bus para su mensajería. Azure Service Bus es un agente de mensajes de integración totalmente administrado que permite que las aplicaciones y los servicios se comuniquen entre sí de forma desacoplada, confiable y asincrónica. Azure Service Bus admite colas individuales, así como *temas* independientes para admitir escenarios de publicador y suscriptor. La aplicación eShopOnContainers aprovecharía temas con Azure Service Bus para admitir la distribución de mensajes de un microservicio a cualquier otro microservicio necesario para reaccionar a un mensaje determinado.

Resiliencia

Una vez implementada en producción, la aplicación eShopOnContainers podría aprovechar las ventajas de varios servicios de Azure disponibles para mejorar su resistencia. La aplicación publica comprobaciones de estado, que se pueden integrar con Application Insights para proporcionar informes y alertas en función de la disponibilidad de la aplicación. Los recursos de Azure también proporcionan registros de diagnóstico que se pueden usar para identificar y corregir errores y problemas de rendimiento. Los registros de recursos proporcionan información detallada sobre cuándo y cómo la aplicación usa distintos recursos de Azure. Obtendrá más información sobre las características de resistencia nativas de la nube en el capítulo 6.

Siguiente anterior

Implementación de eShopOnContainers en Azure

Artículo • 22/05/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

La aplicación eShopOnContainers se puede implementar en varias plataformas de Azure. El enfoque recomendado es implementar la aplicación en Azure Kubernetes Services (AKS). Helm, una herramienta de implementación de Kubernetes, está disponible para reducir la complejidad de la implementación. Opcionalmente, los desarrolladores pueden implementar Azure Dev Spaces para Kubernetes para simplificar su proceso de desarrollo.

Azure Kubernetes Service

Para hospedar eShop en AKS, el primer paso es crear un clúster de AKS. Para ello, puede usar Azure Portal, que le guiará por los pasos necesarios. También puede crear un clúster desde la CLI de Azure, asegurándose de habilitar Role-Based Control de acceso (RBAC) y el enrutamiento de aplicaciones. La documentación de eShopOnContainers detalla los pasos para crear su propio clúster de AKS. Una vez creado, puede acceder al clúster y administrarlo desde el panel de Kubernetes.

Ahora puede implementar la aplicación eShop en el clúster mediante Helm.

Implementación en Azure Kubernetes Service mediante Helm

Helm es una herramienta de administrador de paquetes de aplicaciones que funciona directamente con Kubernetes. Le ayuda a definir, instalar y actualizar aplicaciones de Kubernetes. Aunque las aplicaciones sencillas se pueden implementar en AKS con scripts de la CLI personalizados o archivos de implementación simples, las aplicaciones complejas pueden contener muchos objetos de Kubernetes y beneficiarse de Helm.

Con Helm, las aplicaciones incluyen archivos de configuración basados en texto, denominados gráficos de Helm, que describen mediante declaración la aplicación y la configuración en paquetes de Helm. Los gráficos usan archivos con formato YAML estándar para describir un conjunto relacionado de recursos de Kubernetes. Se versionan junto con el código de la aplicación que describen. Los gráficos de Helm varían de simple a complejo en función de los requisitos de la instalación que describen.

Helm se compone de una herramienta cliente de línea de comandos, que consume gráficos de Helm e inicia comandos en un componente de servidor denominado Tiller. Tiller se comunica con la API de Kubernetes para garantizar el aprovisionamiento correcto de las cargas de trabajo en contenedor. Helm se mantiene mediante Cloud-native Computing Foundation.

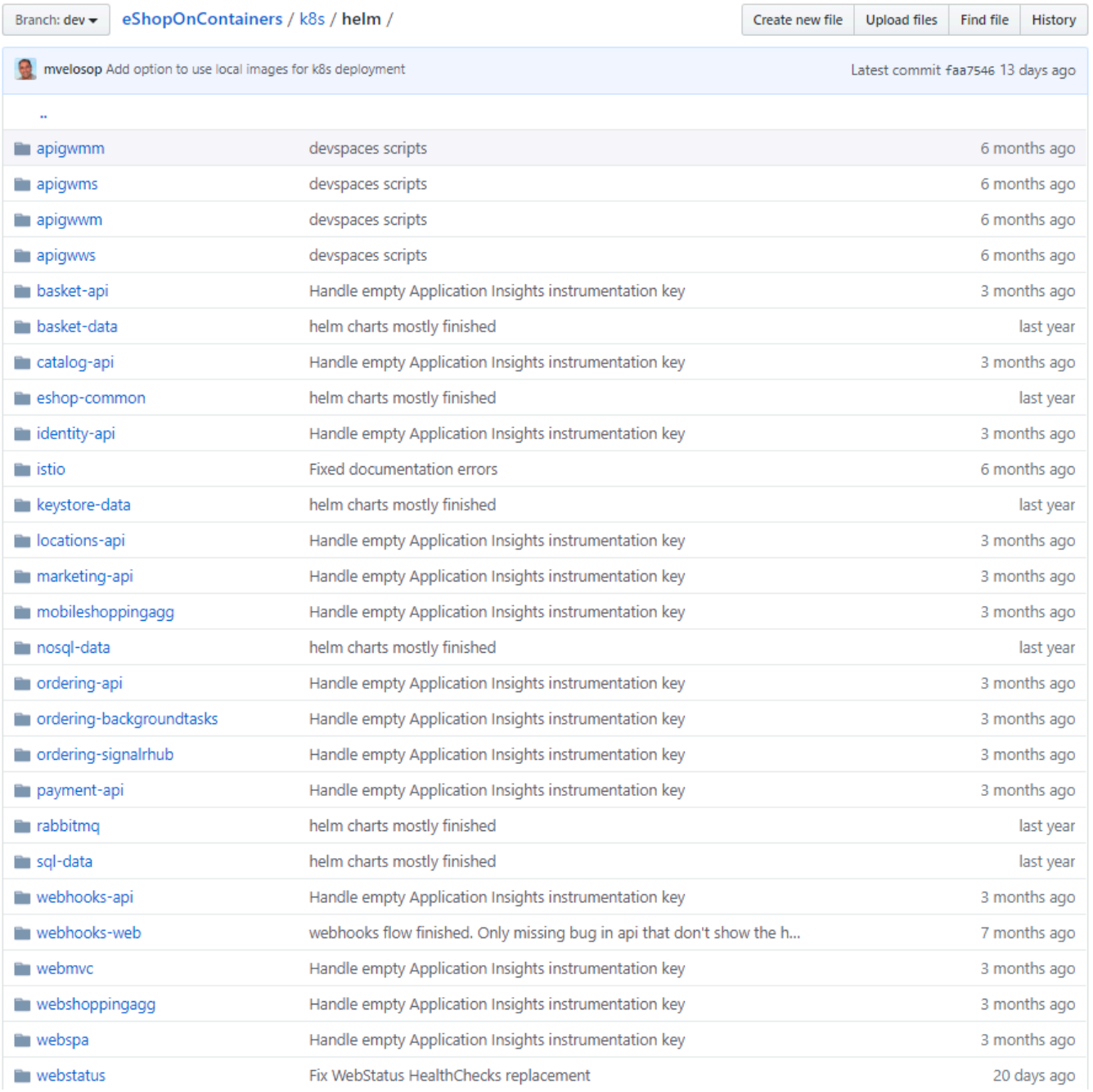
El siguiente archivo yaml presenta una plantilla de Helm:

YAML

```yaml
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.app.svc.marketing }}
  labels:
    app: {{ template "marketing-api.name" . }}
    chart: {{ template "marketing-api.chart" . }}
    release: {{ .Release.Name }}
    heritage: {{ .Release.Service }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: http
      protocol: TCP
      name: http
  selector:
    app: {{ template "marketing-api.name" . }}
    release: {{ .Release.Name }}
```

Observe cómo la plantilla describe un conjunto dinámico de pares clave-valor. Cuando se invoca la plantilla, los valores que se incluyen entre llaves se extraen de otros archivos de configuración basados en YAML.

Encontrará los gráficos de helm de eShopOnContainers en la carpeta /k8s/helm. En la figura 2-6 se muestra cómo se organizan los distintos componentes de la aplicación en una estructura de carpetas usada por Helm para definir e implementar implementaciones administradas.



Branch: dev ▾ | eShopOnContainers / k8s / helm / | Create new file | Upload files | Find file | History

mvelosop Add option to use local images for k8s deployment — Latest commit faa7546 13 days ago

..		
apigwmm	devspaces scripts	6 months ago
apigwms	devspaces scripts	6 months ago
apigwwm	devspaces scripts	6 months ago
apigwws	devspaces scripts	6 months ago
basket-api	Handle empty Application Insights instrumentation key	3 months ago
basket-data	helm charts mostly finished	last year
catalog-api	Handle empty Application Insights instrumentation key	3 months ago
eshop-common	helm charts mostly finished	last year
identity-api	Handle empty Application Insights instrumentation key	3 months ago
istio	Fixed documentation errors	6 months ago
keystore-data	helm charts mostly finished	last year
locations-api	Handle empty Application Insights instrumentation key	3 months ago
marketing-api	Handle empty Application Insights instrumentation key	3 months ago
mobileshoppingagg	Handle empty Application Insights instrumentation key	3 months ago
nosql-data	helm charts mostly finished	last year
ordering-api	Handle empty Application Insights instrumentation key	3 months ago
ordering-backgroundtasks	Handle empty Application Insights instrumentation key	3 months ago
ordering-signalrhub	Handle empty Application Insights instrumentation key	3 months ago
payment-api	Handle empty Application Insights instrumentation key	3 months ago
rabbitmq	helm charts mostly finished	last year
sql-data	helm charts mostly finished	last year
webhooks-api	Handle empty Application Insights instrumentation key	3 months ago
webhooks-web	webhooks flow finished. Only missing bug in api that don't show the h...	7 months ago
webmvc	Handle empty Application Insights instrumentation key	3 months ago
webshoppingagg	Handle empty Application Insights instrumentation key	3 months ago
webspa	Handle empty Application Insights instrumentation key	3 months ago
webstatus	Fix WebStatus HealthChecks replacement	20 days ago

figura 2-6. Carpeta de helm de eShopOnContainers.

Cada componente individual se instala mediante un `helm install` comando . eShop incluye un script de "implementación de todo" que recorre en bucle e instala los componentes mediante sus respectivos gráficos de Helm. El resultado es un proceso repetible, versionado junto con la aplicación en el control de código fuente, que cualquier persona del equipo puede desplegar en un clúster de AKS con un comando en una sola línea de script.

> ⓘ **Nota**

La versión 3 de Helm elimina oficialmente la necesidad del componente de servidor Tiller. Para obtener más información sobre esta mejora, consulte **¿Por qué falta Tiller en Helm 3?**.

Azure Functions y Logic Apps (sin servidor)

El ejemplo eShopOnContainers incluye compatibilidad con el seguimiento de campañas de marketing en línea. Una función de Azure se usa para realizar un seguimiento de los detalles de la campaña de marketing de un identificador de campaña determinado. En lugar de crear un microservicio completo, una única función de Azure es más sencilla y suficiente. Azure Functions tiene un modelo de compilación e implementación sencillo, especialmente cuando está configurado para ejecutarse en Kubernetes. La implementación de la función se crea mediante scripts mediante plantillas de Azure Resource Manager (ARM) y la CLI de Azure. Este servicio de campaña no es orientado al cliente e invoca una sola operación, lo que lo convierte en un excelente candidato para Azure Functions. La función requiere una configuración mínima, incluidos los datos de la cadena de conexión de la base de datos y la configuración del URI base de la imagen. Configure Azure Functions en Azure Portal.

Siguiente anterior

Configuración centralizada

19/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

A diferencia de una aplicación monolítica en la que todo se ejecuta dentro de una sola instancia, una aplicación nativa de la nube consta de servicios independientes distribuidos entre máquinas virtuales, contenedores y regiones geográficas. La administración de opciones de configuración para docenas de servicios interdependientes puede ser difícil. Las copias duplicadas de la configuración en diferentes ubicaciones son propensas a errores y son difíciles de administrar. La configuración centralizada es un requisito fundamental para las aplicaciones nativas de nube distribuidas.

Como se describe en el capítulo 1, las recomendaciones de Twelve-Factor aplicación requieren una separación estricta entre el código y la configuración. La configuración se debe almacenar externamente desde la aplicación y leerla según sea necesario. Almacenar valores de configuración como constantes o valores literales en el código es una infracción. A menudo, muchos servicios de la misma aplicación usan los mismos valores de configuración. Además, debemos admitir los mismos valores en varios entornos, como desarrollo, pruebas y producción. El procedimiento recomendado es almacenarlos en un almacén de configuración centralizado.

La nube de Azure presenta varias opciones excelentes.

Azure App Configuration (Configuración de aplicaciones de Azure)

Azure App Configuration es un servicio de Azure totalmente administrado que almacena valores de configuración no secretos en una ubicación segura y centralizada. Los valores almacenados se pueden compartir entre varios servicios y aplicaciones.

El servicio es fácil de usar y proporciona varias ventajas:

- Representaciones y asignaciones de clave/valor flexibles
- Etiquetado con etiquetas de Azure
- Interfaz de usuario dedicada para la administración
- Cifrado de información confidencial
- Consulta y recuperación por lotes

Azure App Configuration mantiene los cambios realizados en la configuración de clave-valor durante siete días. La característica de instantánea a un momento dado permite reconstruir el historial de una configuración e incluso revertir una implementación con errores.

App Configuration almacena automáticamente en caché cada opción para evitar llamadas excesivas al almacén de configuración. La operación de actualización espera hasta que el valor almacenado en caché de una configuración expira para actualizar esa configuración, incluso cuando su valor cambia en el almacén de configuración. El tiempo de expiración de caché predeterminado es de 30 segundos. Puede invalidar la hora de expiración.

App Configuration cifra todos los valores de configuración en tránsito y en reposo. Los nombres de clave y las etiquetas se usan como índices para recuperar datos de configuración y no se cifran.

Aunque App Configuration proporciona seguridad protegida, Azure Key Vault sigue siendo el mejor lugar para almacenar secretos de aplicación. Key Vault proporciona cifrado de nivel de hardware, directivas de acceso pormenorizadas y operaciones de administración como la rotación de certificados. Puede crear valores de App Configuration que hagan referencia a secretos almacenados en un almacén de claves.

Azure Key Vault

Key Vault es un servicio administrado para almacenar y acceder a secretos de forma segura. Un secreto es todo lo que desea controlar estrechamente el acceso, como claves de API, contraseñas o certificados. Un almacén es un grupo lógico de secretos.

Key Vault reduce en gran medida las posibilidades de que se puedan filtrar por accidente los secretos. Al usar Key Vault, los desarrolladores de aplicaciones ya no necesitan almacenar información de seguridad en su aplicación. Esta práctica elimina la necesidad de almacenar esta información dentro del código. Por ejemplo, es posible que una aplicación tenga que

conectarse a una base de datos. En lugar de almacenar la cadena de conexión en el código de la aplicación, puede almacenarla de forma segura en Key Vault.

Las aplicaciones pueden acceder de manera protegida a la información que necesitan a través de URI. Estos URI permiten que las aplicaciones recuperen versiones específicas de un secreto. No es necesario escribir código personalizado para proteger la información de secretos almacenada en Key Vault.

El acceso a Key Vault requiere una autenticación y autorización adecuadas del autor de la llamada. Normalmente, cada microservicio nativo de la nube usa una combinación ClientId/ClientSecret. Es importante mantener estas credenciales fuera del control de código fuente. Un procedimiento recomendado es establecerlos en el entorno de la aplicación. El acceso directo a Key Vault desde AKS se puede lograr mediante Key Vault FlexVolume.

Configuración en eShop

La aplicación eShopOnContainers incluye archivos de configuración de la aplicación local con cada microservicio. Estos archivos están protegidos en el control de código fuente, pero no incluyen secretos de producción, como cadenas de conexión o claves de API. En producción, la configuración individual se puede sobrescribir con variables de entorno específicas de cada servicio. La inserción de secretos en variables de entorno es una práctica habitual para las aplicaciones hospedadas, pero no proporciona un almacén de configuración central. Para admitir la administración centralizada de las opciones de configuración, cada microservicio incluye una configuración para alternar entre su uso de la configuración local o la configuración de Azure Key Vault.

Referencias

- La arquitectura de eShopOnContainers
- Orquestación de microservicios y aplicaciones de varios contenedores para una alta escalabilidad y disponibilidad
- Azure API Management
- Introducción a Azure SQL Database
- Azure Cache for Redis
- API de Azure Cosmos DB para MongoDB
- Azure Service Bus
- Introducción a Azure Monitor
- eShopOnContainers: Creación de un clúster de Kubernetes en AKS
- eShopOnContainers: Azure Dev Spaces
- Azure Dev Spaces

Siguiente anterior

Escalado de aplicaciones nativas de la nube

17/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Una de las ventajas más frecuentes de pasar a un entorno de hospedaje en la nube es la escalabilidad. Escalabilidad o la posibilidad de que una aplicación acepte una carga de usuario adicional sin poner en peligro el rendimiento de cada usuario. Se consigue con más frecuencia dividiendo una aplicación en pequeños fragmentos que se pueden asignar a cada uno de los recursos que necesiten. Los proveedores de nube permiten una escalabilidad masiva en cualquier momento y en cualquier lugar del mundo.

En este capítulo se describen las tecnologías que permiten que las aplicaciones nativas de la nube se escalen para satisfacer la demanda del usuario. Estas son algunas de esas tecnologías:

- Contenedores
- Orquestadores
- Informática sin servidor

Siguiente anterior

Aprovechamiento de contenedores y orquestadores

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Los contenedores y orquestadores están diseñados para resolver problemas comunes a los enfoques de implementación monolíticos.

Desafíos con implementaciones monolíticas

Tradicionalmente, la mayoría de las aplicaciones se han implementado como una sola unidad. Estas aplicaciones se conocen como un monolito. Este enfoque general de la implementación de aplicaciones como unidades únicas incluso si se componen de varios módulos o ensamblados se conoce como arquitectura monolítica, como se muestra en la figura 3-1.

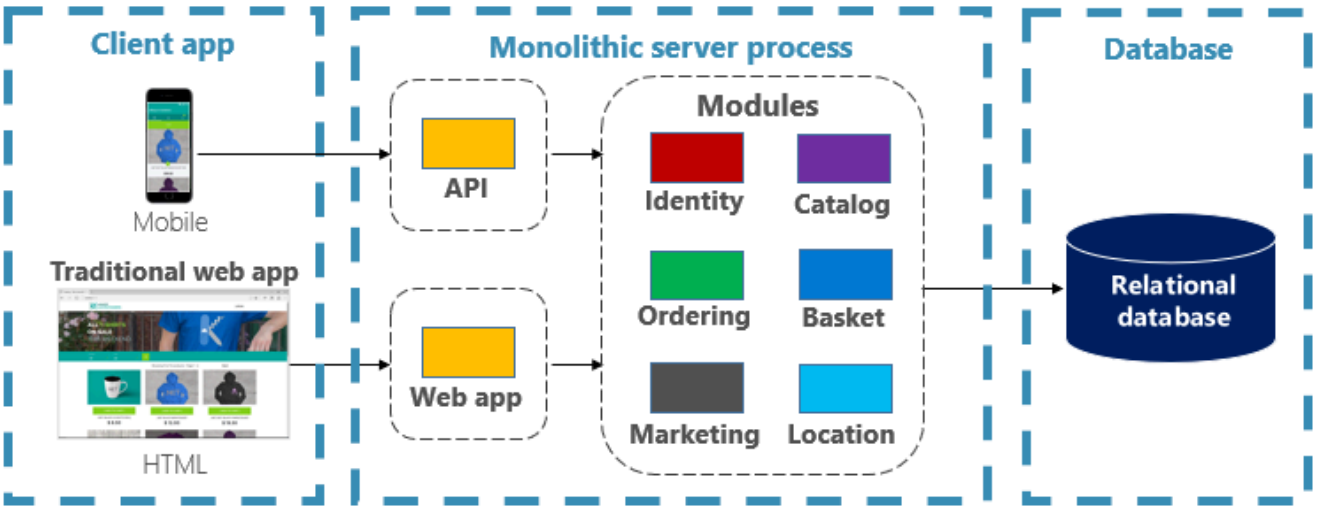


Figura 3-1. Arquitectura monolítica.

Aunque tienen el beneficio de la simplicidad, las arquitecturas monolíticas se enfrentan a muchos desafíos:

Despliegue

Además, requieren un reinicio de la aplicación, que puede afectar temporalmente a la disponibilidad si no se aplican técnicas de tiempo de inactividad cero durante la implementación.

Ampliación

Una aplicación monolítica se hospeda completamente en una sola instancia de máquina, a menudo requiere hardware de alta funcionalidad. Si alguna parte del monolito requiere escalado, se debe implementar otra copia de toda la aplicación en otra máquina. Con un monolito, no se pueden escalar los componentes de la aplicación individualmente, es todo o nada. El escalado de componentes que no requieren el escalado da como resultado un uso de recursos ineficaz y costoso.

Medio ambiente

Normalmente, las aplicaciones monolíticas se implementan en un entorno de hospedaje con un sistema operativo, un entorno de ejecución y dependencias de biblioteca preinstalados. Este entorno no puede coincidir con el que se desarrolló o probó la aplicación. Las incoherencias entre entornos de aplicación son un origen común de problemas para las implementaciones monolíticas.

Acoplamiento

Es probable que una aplicación monolítica experimente un acoplamiento elevado en sus componentes funcionales. Sin límites duros, los cambios del sistema suelen dar lugar a efectos secundarios no deseados y costosos. Las nuevas características y correcciones se vuelven complicadas, lentas y costosas de implementar. Las actualizaciones requieren pruebas exhaustivas. El acoplamiento también dificulta la refactorización de componentes o el intercambio en implementaciones alternativas. Incluso cuando se construye con una separación estricta de cada aspecto, la erosión arquitectónica se hace cada vez más patente a medida que la base de código monolítica se va deteriorando con un sinfín de "casos especiales".

Bloqueo de plataformas

Una aplicación monolítica se construye con una sola pila de tecnología. Aunque ofrece uniformidad, este compromiso puede convertirse en una barrera para la innovación. Las características y componentes nuevos se crearán con la pila actual de la aplicación, incluso cuando las tecnologías más modernas puedan constituir una mejor opción. Un riesgo a largo plazo es que el stack tecnológico se vuelva anticuado y obsoleto. Rediseñar una aplicación completa en una plataforma nueva y más moderna es a la mejor cara y arriesgada.

¿Cuáles son las ventajas de los contenedores y orquestadores?

Hemos introducido contenedores en el capítulo 1. Hemos resaltado cómo Cloud Native Computing Foundation (CNCF) clasifica la contenedorización como primer paso en su mapa de seguimiento deCloud-Native : guía para las empresas que comienzan su recorrido nativo de la nube. En esta sección se describen las ventajas de los contenedores.

Docker es la plataforma de administración de contenedores más popular. Funciona con contenedores en Linux o Windows. Los contenedores proporcionan entornos de aplicación independientes pero reproducibles que se ejecutan de la misma manera en cualquier sistema. Este aspecto hace que sean perfectos para desarrollar y hospedar servicios nativos en la nube. Los contenedores están aislados entre sí. Dos contenedores en el mismo hardware host pueden tener versiones diferentes de software, sin causar conflictos.

Los contenedores se definen mediante archivos simples basados en texto que se convierten en artefactos de proyecto y se comprueban en el control de código fuente. Aunque los servidores completos y las máquinas virtuales requieren un esfuerzo manual para actualizarlos, los contenedores se gestionan fácilmente con control de versiones. Las aplicaciones creadas para ejecutarse en contenedores se pueden desarrollar, probar e implementar mediante herramientas automatizadas como parte de una canalización de compilación.

Los contenedores son inmutables. Una vez definido un contenedor, puede volver a crearlo y ejecutarlo exactamente de la misma manera. Esta inmutabilidad se presta al diseño basado en componentes. Si algunas partes de una aplicación evolucionan de forma diferente a otras, ¿por qué volver a implementar toda la aplicación cuando solo puede implementar las partes que cambian con más frecuencia? Se pueden dividir diferentes características y preocupaciones transversales de una aplicación en unidades independientes. En la figura 3-2 se muestra cómo una aplicación monolítica puede aprovechar los contenedores y microservicios mediante la delegación de determinadas características o funcionalidades. La funcionalidad restante de la propia aplicación también se ha contenedorizado.

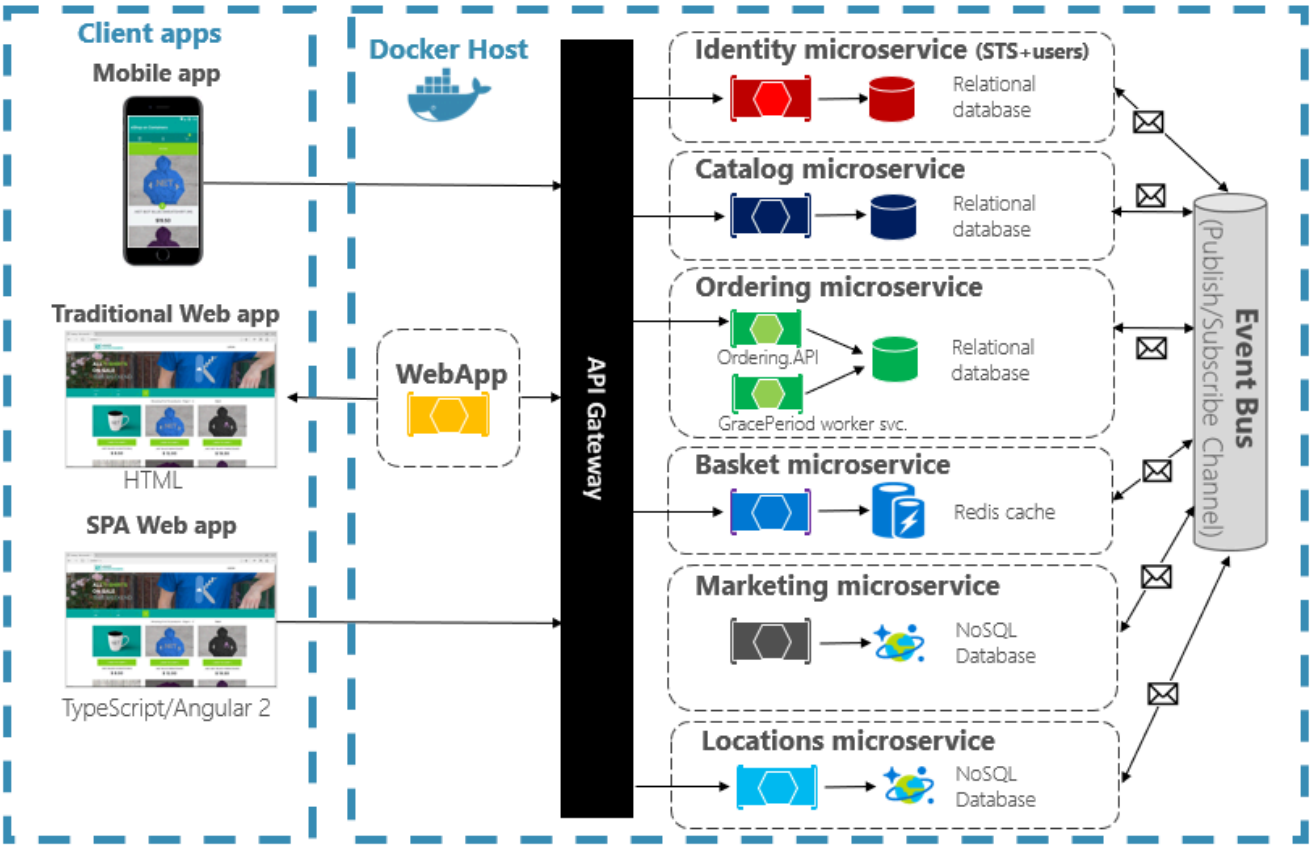


Figura 3-2. Descomponer una aplicación monolítica para adoptar microservicios.

Cada servicio nativo en la nube se compila e implementa en un contenedor independiente. Cada uno puede actualizar según sea necesario. Los servicios individuales se pueden hospedar en nodos con recursos adecuados para cada servicio. El entorno en el que se ejecuta cada servicio es inmutable, compartido entre entornos de desarrollo, pruebas y producción, y con versiones sencillas. El acoplamiento entre diferentes áreas de la aplicación se produce explícitamente como llamadas o mensajes entre servicios, no como dependencias en tiempo de compilación dentro del monolito. También puedes elegir la tecnología que mejor se adapte a una funcionalidad determinada sin necesidad de realizar cambios en el resto de la aplicación.

Los servicios en contenedores requieren administración automatizada. No sería factible administrar manualmente un gran conjunto de contenedores implementados de forma independiente. Por ejemplo, tenga en cuenta las siguientes tareas:

- ¿Cómo se aprovisionarán las instancias de contenedor en un clúster con muchas máquinas?
- Una vez implementado, ¿cómo detectarán los contenedores y se comunicarán entre sí?
- ¿Cómo se puede aumentar o reducir el número de contenedores a demanda?
- ¿Cómo se supervisa el estado de cada contenedor?
- ¿Cómo protege un contenedor frente a errores de hardware y software?
- ¿Cómo se actualizan los contenedores de una aplicación activa sin tiempo de inactividad?

Los orquestadores de contenedores abordan y automatizan estos y otros problemas.

En el sistema ecológico nativo de la nube, Kubernetes se ha convertido en el orquestador de contenedores de facto. Es una plataforma de código abierto administrada por Cloud Native Computing Foundation (CNCF). Kubernetes automatiza la implementación, el escalado y las preocupaciones operativas de las cargas de trabajo en contenedores en un clúster de máquinas. Sin embargo, la instalación y administración de Kubernetes es notoriamente compleja.

Un enfoque mucho mejor es aprovechar Kubernetes como servicio administrado de un proveedor de nube. La nube de Azure incluye una plataforma de Kubernetes totalmente administrada titulada Azure Kubernetes Service (AKS). AKS abstrae la complejidad y la sobrecarga operativa de la administración de Kubernetes. Consume Kubernetes como servicio en la nube; Microsoft asume la responsabilidad de administrarla y apoyarla. AKS también se integra estrechamente con otros servicios y herramientas de desarrollo de Azure.

AKS es una tecnología basada en clústeres. Un grupo de máquinas virtuales federadas o nodos se implementa en la nube de Azure. Juntos forman un entorno de alta disponibilidad o un clúster. El clúster aparece como una entidad única e integrada sin interrupciones en su aplicación nativa de la nube. En segundo plano, AKS implementa los servicios en contenedor en estos nodos siguiendo una estrategia predefinida que distribuye uniformemente la carga.

¿Cuáles son las ventajas de escalado?

Los servicios basados en contenedores pueden aprovechar las ventajas de escalado proporcionadas por herramientas de orquestación como Kubernetes. Por diseño, los contenedores solo saben de sí mismos. Una vez que tenga varios contenedores que necesiten trabajar juntos, debe organizarlos en un nivel superior. Organizar un gran número de contenedores y sus dependencias compartidas, como la configuración de una red... ¡ahí es donde las herramientas de orquestación entran en juego para salvarnos los muebles! Kubernetes crea una capa de abstracción sobre grupos de contenedores y las organiza en *pods*. Los pods se ejecutan en máquinas de trabajo denominadas *nodos*. Esta estructura organizada se conoce como *un clúster*. En la figura 3-3 se muestran los distintos componentes de un clúster de Kubernetes.

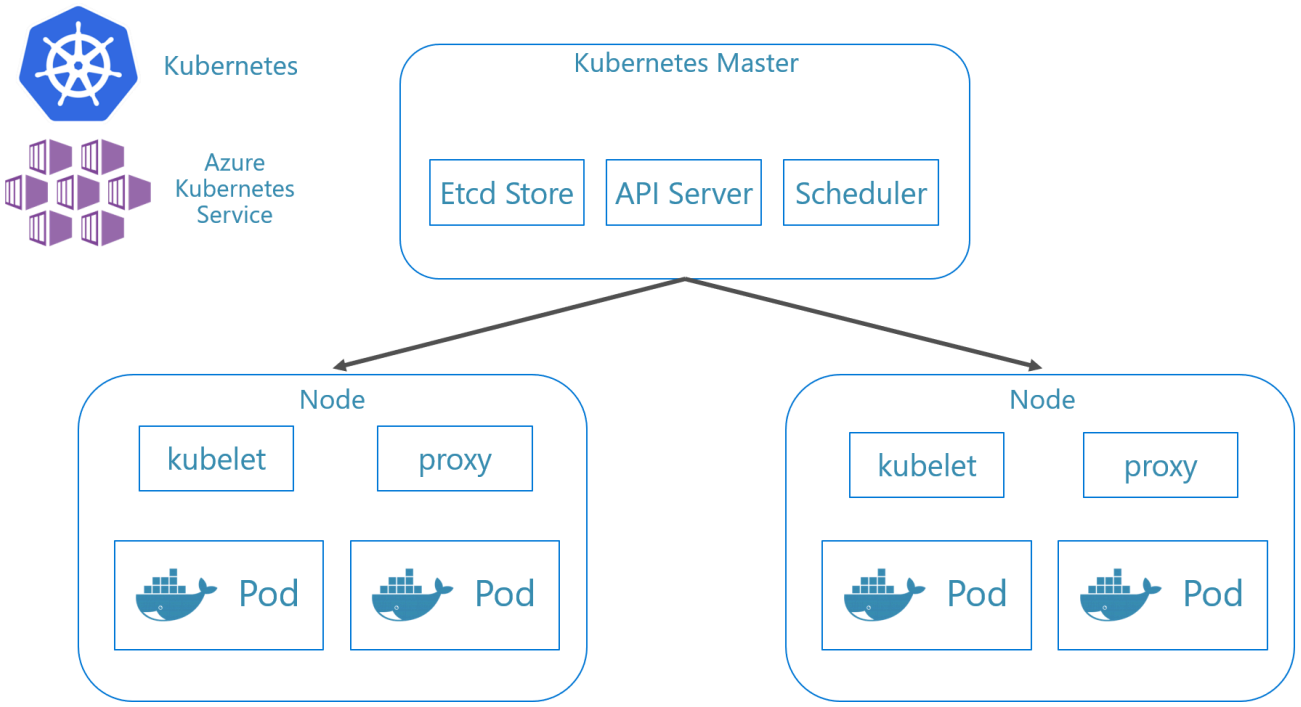


Figura 3-3. Componentes del clúster de Kubernetes.

El escalado de cargas de trabajo en contenedores es una característica clave de los orquestadores de contenedores. AKS admite el escalado automático entre dos dimensiones: instancias de contenedor y nodos de proceso. Juntos proporcionan a AKS la capacidad de responder de forma rápida y eficaz a los picos de demanda y agregar recursos adicionales. Analizaremos el escalado en AKS más adelante en este capítulo.

Declarativo frente a imperativo

Kubernetes admite la configuración declarativa e imperativa. El enfoque imperativo implica ejecutar varios comandos que indican a Kubernetes qué hacer cada paso del camino. Ejecute esta imagen. Elimina este pod. Expón este puerto. Con el enfoque declarativo, se crea un archivo de configuración, denominado manifiesto, para describir lo que desea en lugar de lo que debe hacer. Kubernetes lee el manifiesto y transforma el estado final deseado en estado final real.

Los comandos imperativos son excelentes para el aprendizaje y la experimentación interactiva. pero lo más conveniente es crear archivos de manifiesto de Kubernetes de forma declarativa para adoptar el uso de una infraestructura como enfoque de código, lo que dará lugar a implementaciones confiables y repetibles. El archivo de manifiesto se convierte en un elemento del proyecto y se usa en el flujo de trabajo de CI/CD para automatizar las implementaciones de Kubernetes.

Si ya ha configurado el clúster mediante comandos imperativos, puede exportar un manifiesto declarativo mediante `kubectl get svc SERVICENAME -o yaml > service.yaml`. Este comando genera un manifiesto similar al que se muestra a continuación:

YAML

```yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2019-09-13T13:58:47Z"
  labels:
    component: apiserver
    provider: kubernetes
  name: kubernetes
  namespace: default
  resourceVersion: "153"
  selfLink: /api/v1/namespaces/default/services/kubernetes
  uid: 9b1fac62-d62e-11e9-8968-00155d38010d
spec:
  clusterIP: 10.96.0.1
  ports:
  - name: https
    port: 443
    protocol: TCP
    targetPort: 6443
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

Al usar la configuración declarativa, puede obtener una vista previa de los cambios que se realizarán antes de confirmarlos mediante `kubectl diff -f FOLDERNAME` en la carpeta donde se encuentran los archivos de configuración. Una vez que esté seguro de que desea aplicar los cambios, ejecute `kubectl apply -f FOLDERNAME`. Agregue `-R` para procesar recursivamente una jerarquía de carpetas.

También puede usar la configuración declarativa con otras características de Kubernetes, una de las cuales son implementaciones. Las implementaciones declarativas ayudan a administrar versiones, actualizaciones y escalado. Indican al controlador de implementación de Kubernetes cómo implementar nuevos cambios, escalar horizontalmente o revertir a una revisión anterior. Si un clúster es inestable, una implementación declarativa devolverá automáticamente el clúster a un estado deseado. Por ejemplo, si un nodo se bloquea, el mecanismo de implementación desplegará nuevamente un reemplazo para alcanzar el estado deseado.

El uso de la configuración declarativa permite representar la infraestructura como código que se puede comprobar y versionar junto con el código de la aplicación. Proporciona un mejor control de los cambios y una mejor compatibilidad con la implementación continua mediante una canalización de tipo "crear e implementar".

¿Qué escenarios son ideales para usar contenedores y orquestadores?

Los escenarios siguientes son ideales para usar contenedores y orquestadores.

Aplicaciones que requieren un alto tiempo de actividad y escalabilidad

Las aplicaciones individuales que tienen requisitos de alto tiempo de actividad y escalabilidad son candidatas ideales para las arquitecturas nativas de la nube mediante microservicios, contenedores y orquestadores. Se pueden desarrollar en contenedores, probarse en entornos con versiones e implementarse en producción sin tiempo de inactividad. El uso de clústeres de Kubernetes garantiza que estas aplicaciones también se pueden escalar a petición y recuperarse automáticamente de errores de nodo.

Gran número de aplicaciones

Las organizaciones que implementan y mantienen un gran número de aplicaciones se benefician de contenedores y orquestadores. El esfuerzo inicial de configurar entornos en contenedor y clústeres de Kubernetes es principalmente un costo fijo. La implementación, el mantenimiento y la actualización de aplicaciones individuales tiene un costo que varía con el número de aplicaciones. Además de algunas aplicaciones, la complejidad de mantener las aplicaciones personalizadas manualmente supera el costo de implementar una solución mediante contenedores y orquestadores.

¿Cuándo debería usted evitar el uso de contenedores y orquestadores?

Si no puede construir su aplicación siguiendo los principios de Twelve-Factor App, debe considerar evitar el uso de contenedores y orquestadores. En estos casos, considere una plataforma de hospedaje basada en máquinas virtuales o, posiblemente, algún sistema híbrido. Siempre puedes desglosar ciertos elementos de funcionalidad en contenedores independientes o incluso funciones sin servidor.

Recursos de desarrollo

En esta sección se muestra una breve lista de recursos de desarrollo que pueden ayudarle a empezar a usar contenedores y orquestadores para la siguiente aplicación. Si busca

instrucciones sobre cómo diseñar la aplicación de arquitectura de microservicios nativos en la nube, lea el complemento de este libro, Microservicios de .NET: Arquitectura para aplicaciones .NET en contenedor.

Desarrollo de Kubernetes local

Las implementaciones de Kubernetes proporcionan un gran valor en entornos de producción, pero también se pueden ejecutar localmente en la máquina de desarrollo. Aunque puede trabajar en microservicios individuales de forma independiente, puede haber ocasiones en las que tendrá que ejecutar todo el sistema localmente, igual que se ejecutará cuando se implemente en producción. Hay varias herramientas que pueden ayudar: Minikube y Docker Desktop. Visual Studio también proporciona herramientas para el desarrollo de Docker.

Minikube

¿Qué es Minikube? El proyecto Minikube dice que "Minikube implementa un clúster de Kubernetes local en macOS, Linux y Windows". Sus objetivos principales son "ser la mejor herramienta para el desarrollo de aplicaciones locales de Kubernetes y admitir todas las características de Kubernetes que se ajusten". La instalación de Minikube es independiente de Docker, pero Minikube admite hipervisores diferentes que los compatibles con Docker Desktop. Las siguientes características de Kubernetes son compatibles actualmente con Minikube:

- DNS (Sistema de Nombres de Dominio)
- NodePorts
- ConfigMaps y secretos
- Tableros de control
- Entornos de ejecución de contenedor: Docker, rkt, CRI-O y containerd
- Habilitar la interfaz de red para contenedores (CNI)
- Entrada

Después de instalar Minikube, puede empezar a usarlo rápidamente mediante la ejecución del `minikube start` comando , que descarga una imagen e inicia el clúster de Kubernetes local. Una vez iniciado el clúster, interactúe con él mediante los comandos estándar de Kubernetes `kubectl` .

Docker Desktop

También puede trabajar con Kubernetes directamente desde Docker Desktop en Windows. Es la única opción si usa contenedores de Windows y también es una opción excelente para los

contenedores que no son de Windows. En la figura 3-4 se muestra cómo habilitar la compatibilidad con Kubernetes local al ejecutar Docker Desktop.

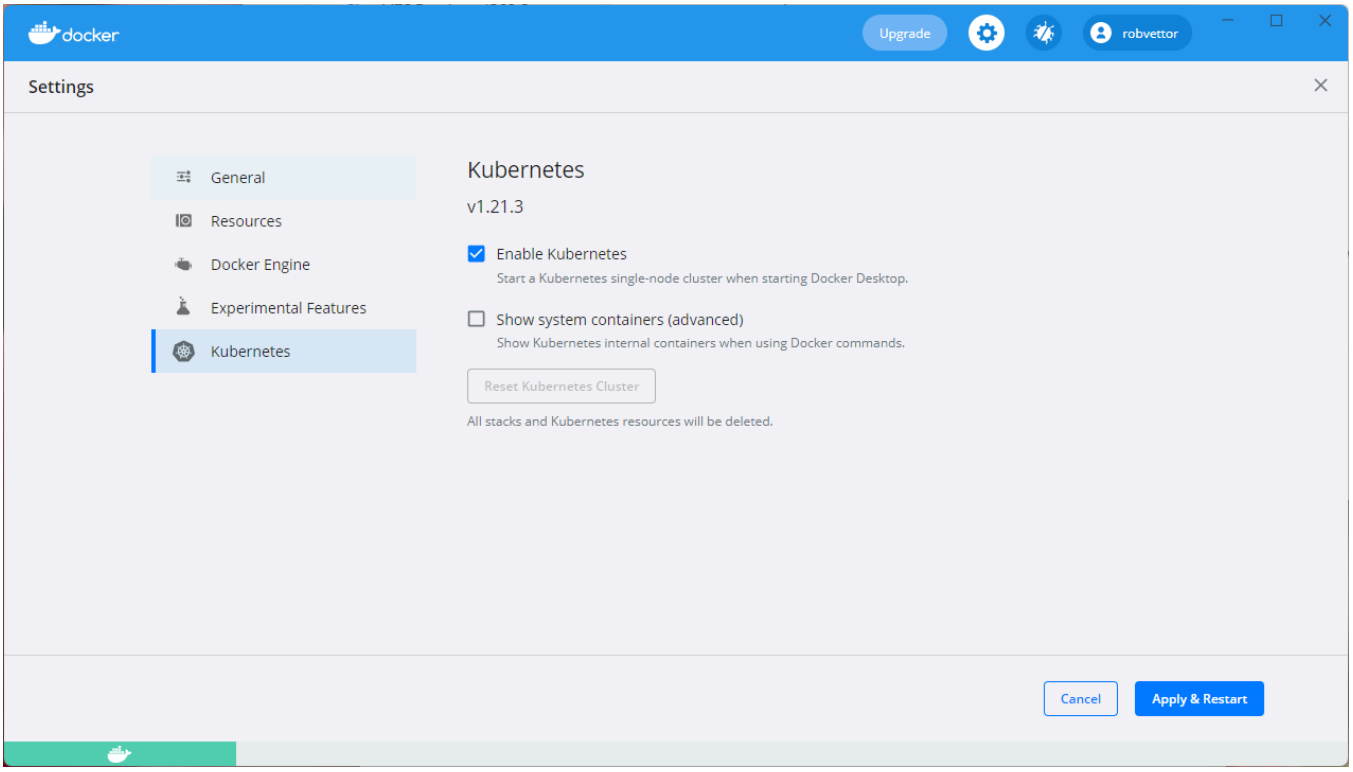


Figura 3-4. Configuración de Kubernetes en Docker Desktop.

Docker Desktop es la herramienta más popular para configurar y ejecutar aplicaciones en contenedores localmente. Cuando trabajas con Docker Desktop, tienes la posibilidad de desarrollar localmente con el mismo conjunto preciso de imágenes de contenedor de Docker que implementarás en producción. Docker Desktop está diseñado para "compilar, probar y enviar" aplicaciones en contenedores localmente. Admite contenedores de Linux y Windows. Una vez insertadas las imágenes en un registro de imágenes, como Azure Container Registry o Docker Hub, AKS puede extraerlas e implementarlas en producción.

Herramientas de Docker de Visual Studio

Visual Studio admite el desarrollo de Docker para aplicaciones basadas en web. Al crear una nueva aplicación ASP.NET Core, tiene la opción de configurarla con compatibilidad con Docker, como se muestra en la figura 3-5.

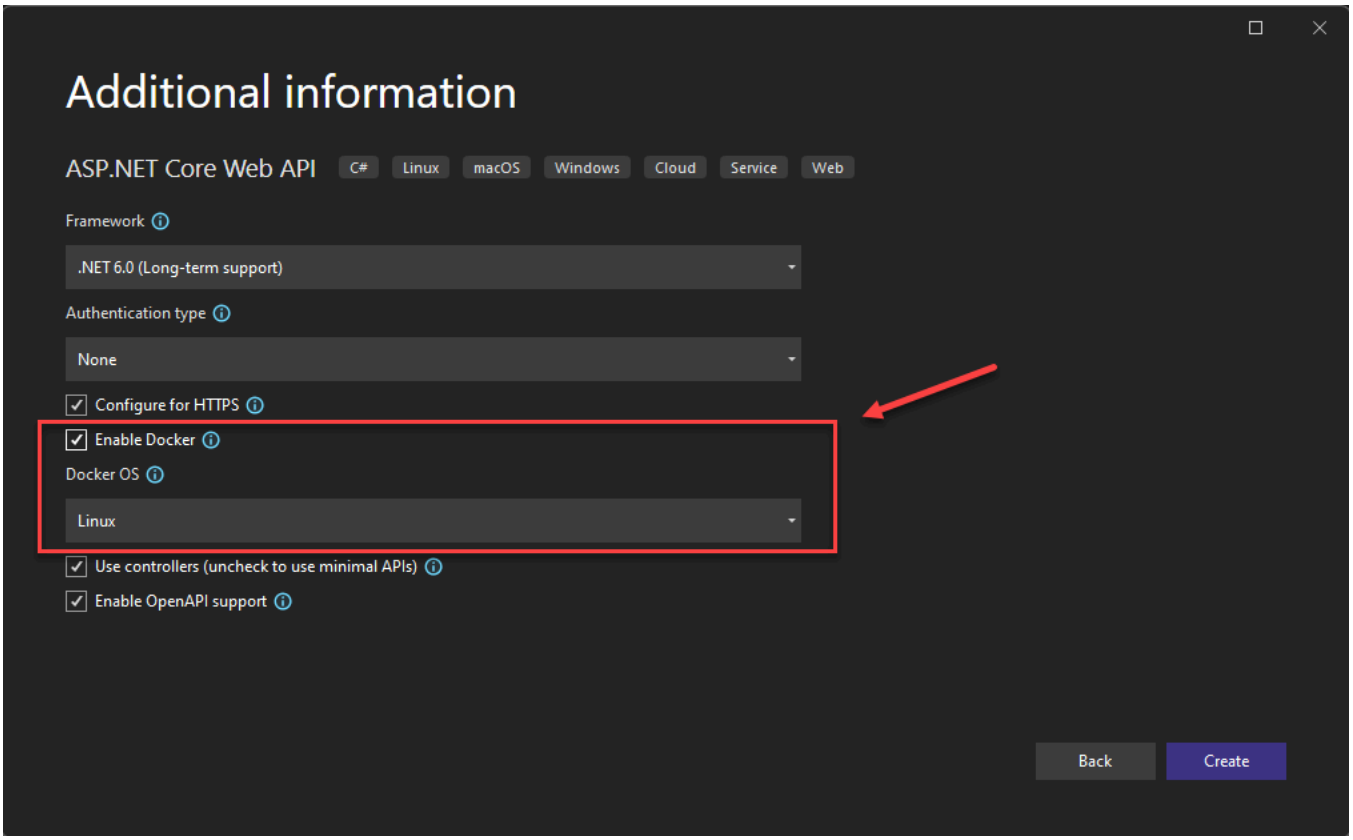


Figura 3-5. Habilitar la compatibilidad con Docker en Visual Studio

Cuando se selecciona esta opción, el proyecto se crea con un `Dockerfile` en su raíz, que se puede usar para crear y alojar la aplicación en un contenedor de Docker. En la figura 3-6 se muestra un dockerfile de ejemplo.

Dockerfile

```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["eShopWeb/eShopWeb.csproj", "eShopWeb/"]
RUN dotnet restore "eShopWeb/eShopWeb.csproj"
COPY . .
WORKDIR "/src/eShopWeb"
RUN dotnet build "eShopWeb.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "eShopWeb.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "eShopWeb.dll"]
```

Figura 3-6. Dockerfile generado por Visual Studio

Una vez agregada la compatibilidad, puede ejecutar la aplicación en un contenedor de Docker en Visual Studio. En la figura 3-7 se muestran las distintas opciones de ejecución disponibles en un nuevo proyecto de ASP.NET Core creado con la compatibilidad con Docker agregada.

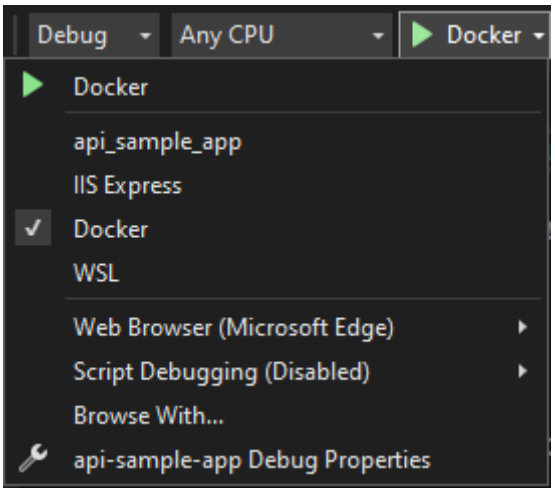


Figura 3-7. Opciones de ejecución de Docker de Visual Studio

Además, en cualquier momento puede agregar compatibilidad con Docker a una aplicación ASP.NET Core existente. En el Explorador de soluciones de Visual Studio, haga clic con el botón derecho en el proyecto y seleccione **Agregar**>**compatibilidad con Docker**, como se muestra en la figura 3-8.

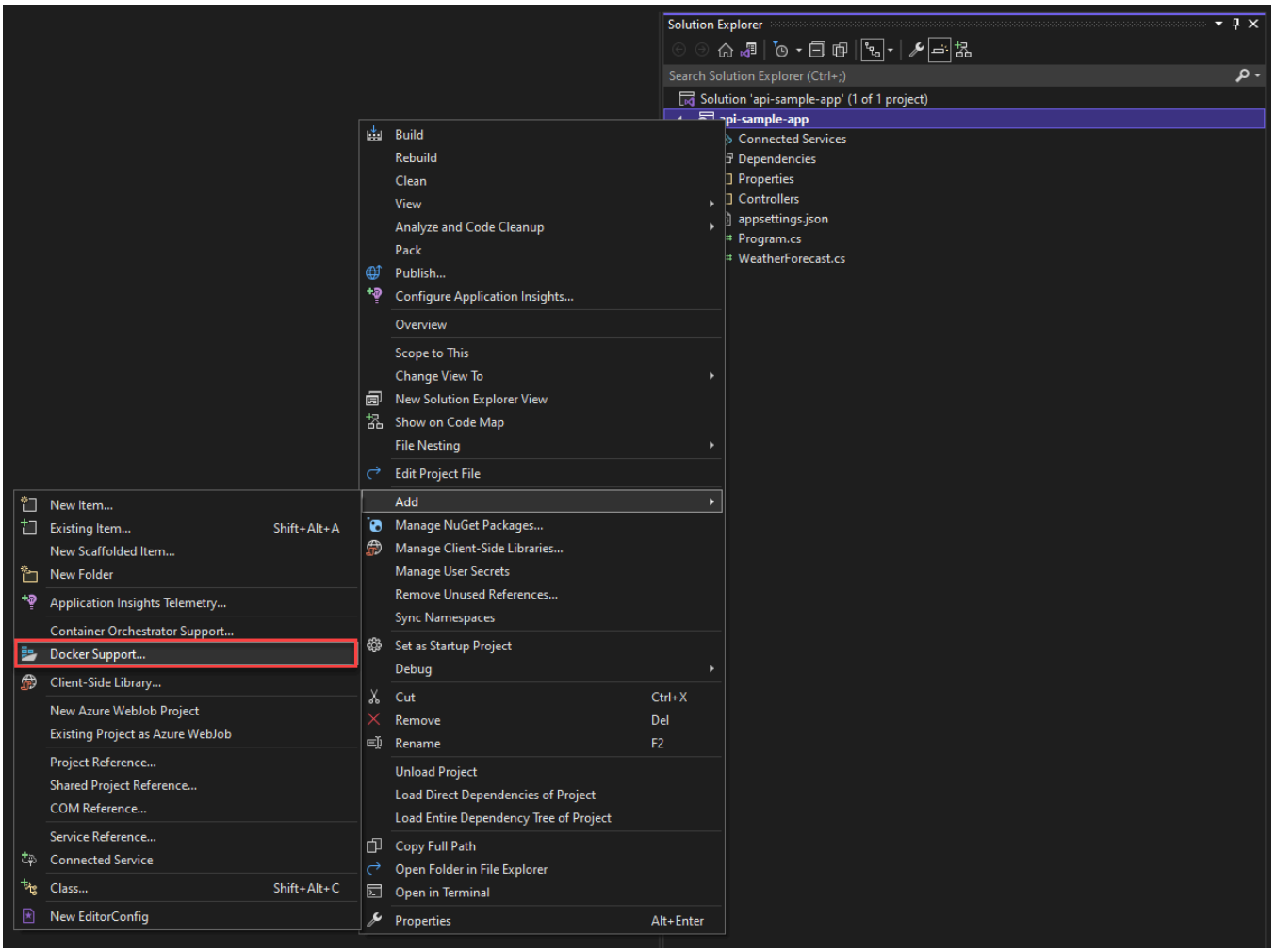


Figura 3-8. Adición de compatibilidad con Docker a Visual Studio

Herramientas Docker de Visual Studio Code

Hay muchas extensiones disponibles para Visual Studio Code que admiten el desarrollo de Docker.

Microsoft proporciona la extensión Docker para Visual Studio Code. Esta extensión simplifica el proceso de agregar compatibilidad con contenedores a las aplicaciones. Estructura los archivos necesarios, construye imágenes de Docker y le permite depurar su aplicación dentro de un contenedor. La extensión incluye un explorador visual que facilita la ejecución de acciones en contenedores e imágenes, como iniciar, detener, inspeccionar, quitar y mucho más. La extensión también admite Docker Compose, lo que le permite administrar varios contenedores en ejecución como una sola unidad.

Siguiente anterior

Aprovechamiento de funciones sin servidor

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

En el espectro que va desde la gestión de máquinas físicas hasta el aprovechamiento de las capacidades de la nube, la computación sin servidor se encuentra en el extremo final. Su única responsabilidad es el código y solo paga cuando se ejecuta el código. Azure Functions proporciona una manera de crear funcionalidades sin servidor en las aplicaciones nativas de la nube.

¿Qué es la computación sin servidores?

Sin servidor es un modelo de servicio relativamente nuevo de la informática en la nube. No significa que los servidores sean opcionales: el código todavía se ejecuta en un servidor en algún lugar. La distinción es que el equipo de aplicaciones ya no se preocupa por la administración de la infraestructura del servidor. En su lugar, el proveedor en la nube posee esta responsabilidad. El equipo de desarrollo aumenta su productividad al ofrecer soluciones empresariales a los clientes, no a la fontanería.

La informática sin servidor usa contenedores sin estado desencadenados por eventos para hospedar los servicios. Pueden aumentar y reducir la escala para satisfacer la demanda según se necesite. Las plataformas sin servidor, como Azure Functions, tienen una estrecha integración con otros servicios de Azure, como colas, eventos y almacenamiento.

¿Qué desafíos resuelven los sin servidor?

Las plataformas sin servidor abordan muchos problemas costosos y lentos:

- Compra de máquinas y licencias de software
- Carcasa, protección, configuración y mantenimiento de las máquinas y sus requisitos de red, energía y A/C
- Instalación de parches y actualización de sistemas operativos y software
- Configuración de servidores web o servicios de máquina para hospedar software de aplicación
- Configuración del software de aplicación dentro de su plataforma

Muchas empresas asignan grandes presupuestos para respaldar la infraestructura de hardware. El traslado a la nube puede ayudar a reducir estos costos; cambiar las aplicaciones a sin servidor puede ayudar a eliminarlas.

¿Cuál es la diferencia entre un microservicio y una función sin servidor?

Normalmente, un microservicio encapsula una funcionalidad empresarial, como un carro de la compra para un sitio de comercio electrónico en línea. Expone varias operaciones que permiten a un usuario administrar su experiencia de compra. Sin embargo, una función es un pequeño bloque ligero de código que ejecuta una operación de un solo propósito en respuesta a un evento. Normalmente, los microservicios se construyen para responder a las solicitudes, a menudo desde una interfaz. Las solicitudes pueden basarse en REST HTTP o gRPC. Los servicios sin servidor responden a eventos. Su arquitectura controlada por eventos es ideal para procesar tareas en segundo plano y de ejecución corta.

¿Qué escenarios son adecuados para los sin servidor?

La computación sin servidor expone funciones individuales de ejecución corta que se invocan como respuesta a un evento. Esto hace que sean ideales para procesar tareas en segundo plano.

Es posible que una aplicación tenga que enviar un correo electrónico como paso en un flujo de trabajo. En lugar de enviar la notificación como parte de una solicitud de microservicio, coloque los detalles del mensaje en una cola. Una función de Azure puede quitar de la cola el mensaje y enviar el correo electrónico de forma asincrónica. Si lo hace, podría mejorar el rendimiento y la escalabilidad del microservicio. Es posible implementar la nivelación de carga

basada en colas para evitar cuellos de botella relacionados con el envío de correos electrónicos. Además, este servicio independiente podría reutilizarse como una utilidad en muchas aplicaciones diferentes.

La mensajería asincrónica de colas y temas es un patrón común para desencadenar funciones sin servidor. Sin embargo, Azure Functions también pueden ser desencadenadas por otros eventos, como cambios en Azure Blob Storage. Un servicio que admita cargas de imágenes podría tener una función de Azure responsable de optimizar el tamaño de la imagen. La función se puede desencadenar directamente mediante inserciones en Azure Blob Storage, lo que mantiene la complejidad fuera de las operaciones de microservicios.

Muchos servicios tienen procesos de ejecución prolongada como parte de sus flujos de trabajo. A menudo, estas tareas se realizan como parte de la interacción del usuario con la aplicación. Estas tareas pueden obligar al usuario a esperar, lo que afecta negativamente a su experiencia. La informática sin servidor proporciona una excelente manera de mover tareas más lentas fuera del bucle de interacción del usuario. Estas tareas se pueden escalar con demanda sin necesidad de que toda la aplicación se escale.

¿Cuándo debe evitar el uso sin servidor?

Las soluciones sin servidor aprovisionan y escalan bajo demanda. Cuando se invoca una nueva instancia, los arranques en frío son una incidencia común. Un arranque en frío es el período de tiempo que se tarda en aprovisionar esta instancia. Normalmente, este retraso puede ser de unos segundos, pero puede ser más largo dependiendo de varios factores. Una vez aprovisionado, se mantiene activa una sola instancia siempre y cuando reciba solicitudes periódicas. Sin embargo, si se llama a un servicio con menos frecuencia, Azure puede quitarlo de la memoria y requerir un inicio en frío cuando se vuelva a invocar. Los arranques en frío también son necesarios cuando una función se escala horizontalmente a una nueva instancia.

En la figura 3-9 se muestra un patrón de arranque en frío. Observe los pasos adicionales necesarios cuando la aplicación está en frío.

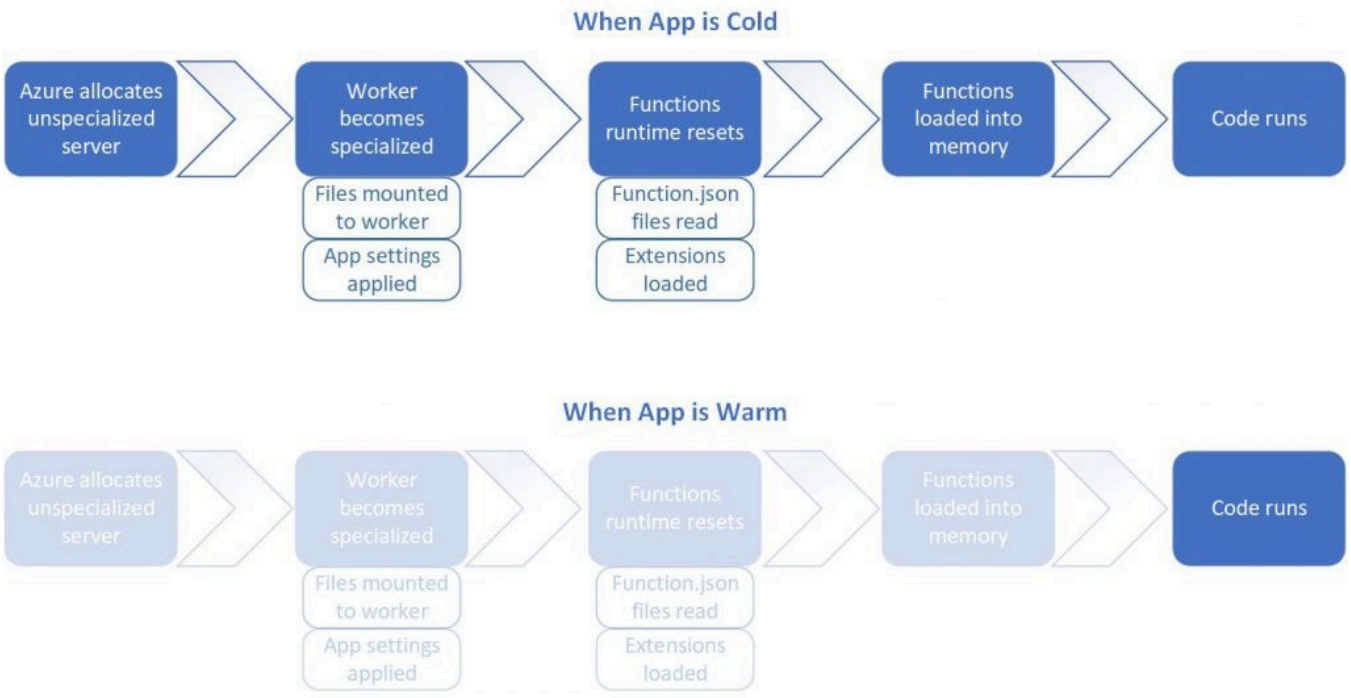


Figura 3-9. Arranque en frío frente a arranque en caliente.

Para evitar los arranques en frío por completo, puede cambiar de un plan de consumo a un plan dedicado. También puede configurar una o varias instancias precalentadas con la actualización del plan Premium. En estos casos, cuando necesite agregar otra instancia, ya está lista para empezar. Estas opciones pueden ayudar a mitigar el problema de arranque en frío asociado a la informática sin servidor.

Los proveedores de nube facturan por servicios sin servidor en función del tiempo de ejecución de computación y la memoria consumida. Las operaciones de larga duración o las cargas de trabajo de consumo elevado de memoria no siempre son las mejores candidatas para sin servidor. Las funciones sin servidor favorecen pequeños fragmentos de trabajo que se pueden completar rápidamente. La mayoría de las plataformas sin servidor requieren que las funciones individuales se completen en unos minutos. Azure Functions tiene como valor predeterminado una duración de tiempo de espera de 5 minutos, que se puede configurar hasta 10 minutos. El plan Premium de Azure Functions también puede mitigar este problema, ya que el tiempo de espera predeterminado es de 30 minutos con un límite superior no limitado que se puede configurar. El tiempo de cómputo no es el tiempo del calendario. Las funciones más avanzadas que usan el marco de Azure Durable Functions pueden pausar la ejecución durante varios días. La facturación se basa en el tiempo de ejecución real, cuando la función se reactiva y reanuda el procesamiento.

Por último, aprovechar Azure Functions para las tareas de aplicación agrega complejidad. Es aconsejable diseñar primero la aplicación con un diseño modular y acoplado flexiblemente. A continuación, identifique si hay ventajas sin servidor que ofrezcan que justifican la complejidad adicional.

Siguiente anterior

Combinación de los enfoques sin servidor y los contenedores

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico "Architecting Cloud Native .NET Applications for Azure" (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Las aplicaciones nativas de nube suelen implementar servicios que hacen uso de los contenedores y la orquestación. A menudo, existe la posibilidad de exponer algunos de los servicios de la aplicación como Azure Functions. Sin embargo, con una aplicación nativa de nube implementada en Kubernetes, sería bueno sacar partido de Azure Functions dentro de ese mismo conjunto de herramientas. Afortunadamente, Azure Functions se puede encapsular en contenedores Docker e implementar con los mismos procesos y herramientas que el resto de la aplicación basada en Kubernetes.

¿Cuándo tiene sentido usar contenedores con un enfoque sin servidor?

La función de Azure Functions no tiene ningún conocimiento de la plataforma en la que se implementa. En algunos escenarios, es posible que existan requisitos específicos y sea necesario personalizar el entorno en el que se va a ejecutar el código de la función. Necesitará una imagen personalizada que admita dependencias o una configuración que no sea compatible con la imagen predeterminada. En estos casos, tiene sentido implementar la función en un contenedor Docker personalizado.

¿Cuándo se debe evitar el uso de contenedores con Azure Functions?

Si se quiere usar la facturación de consumo, la función no se puede ejecutar en un contenedor. Además, si implementa la función en un clúster de Kubernetes, dejará de beneficiarse del escalado integrado proporcionado por Azure Functions. Deberá usar las características de escalado de Kubernetes, descritas anteriormente en este capítulo.

Cómo combinar contenedores Docker y sin servidor

Para encapsular una función de Azure Functions en un contenedor Docker, instale Azure Functions Core Tools y ejecute el siguiente comando:

Consola

```
func init ProjectName --worker-runtime dotnet --docker
```

Cuando el proyecto se cree, incluirá un Dockerfile y el tiempo de ejecución de trabajo configurado en `dotnet`. Ahora, puede crear y probar la función localmente. Créela y ejecútela con los comandos `docker build` y `docker run`. Para conocer los pasos detallados para empezar a crear funciones de Azure Functions con compatibilidad con Docker, vea el tutorial Creación de una función en Linux mediante una imagen personalizada.

Cómo combinar un enfoque sin servidor y Kubernetes con KEDA

En este capítulo hemos visto que la plataforma de Azure Functions escala horizontalmente de manera automática para satisfacer cualquier demanda. Sin embargo, cuando se implementan funciones contenedorizadas en AKS, la funcionalidad de escalado integrada se pierde. Esto se subsana con Kubernetes-based Event-Driven Autoscaling (KEDA), que permite un escalado automático avanzado de `event-driven Kubernetes workloads,`, incluidas las funciones contenedorizadas.

KEDA proporciona una funcionalidad de escalado basada en eventos al entorno de ejecución de Functions en un contenedor Docker. KEDA puede escalar desde cero instancias (no hay eventos) a `n instances`, en función de la carga. Permite el escalado automático mediante la exposición de métricas personalizadas al escalador automático

de Kubernetes (HorizontalPodAutoscaler). El uso de contenedores de Functions con KEDA hace posible replicar las capacidades de la función sin servidor en cualquier clúster de Kubernetes.

Cabe destacar que el proyecto KEDA está administrado actualmente por la Cloud Native Computing Foundation (CNCF).

Anterior Siguiente

Implementación de contenedores en Azure

19/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Hemos analizado los contenedores de este capítulo y en el capítulo 1. Hemos visto que los contenedores proporcionan muchas ventajas a las aplicaciones nativas de la nube, incluida la portabilidad. En la nube de Azure, puede implementar los mismos servicios en contenedores en entornos de ensayo y producción. Azure proporciona varias opciones para hospedar estas cargas de trabajo en contenedor:

- Azure Kubernetes Services (AKS)
- Azure Container Instance (ACI)
- Azure Aplicaciones Web para Contenedores

Azure Container Registry (Registro de Contenedores de Azure)

Al incluir en contenedores un microservicio, primero se compila un contenedor "image". La imagen es una representación binaria del código de servicio, las dependencias y el entorno de ejecución. Aunque puede crear manualmente una imagen mediante el `Docker Build` comando desde la API de Docker, un mejor enfoque es crearla como parte de un proceso de compilación automatizado.

Una vez creadas, las imágenes de contenedor se almacenan en registros de contenedor. Permiten compilar, almacenar y administrar imágenes de contenedor. Hay muchos registros

disponibles, tanto públicos como privados. Azure Container Registry (ACR) es un servicio de registro de contenedor totalmente administrado en la nube de Azure. Conserva las imágenes dentro de la red de Azure, lo que reduce el tiempo para implementarlas en los hosts de contenedor de Azure. También puede protegerlos mediante los mismos procedimientos de seguridad e identidad que se usan para otros recursos de Azure.

Puede crear una instancia de Azure Container Registry mediante Azure Portal, la CLI de Azure o las herramientas de PowerShell. La creación de un registro en Azure es sencilla. Requiere una suscripción de Azure, un grupo de recursos y un nombre único. En la figura 3-10 se muestran las opciones básicas para crear un registro, que se hospedará en `registryname.azurecr.io`.

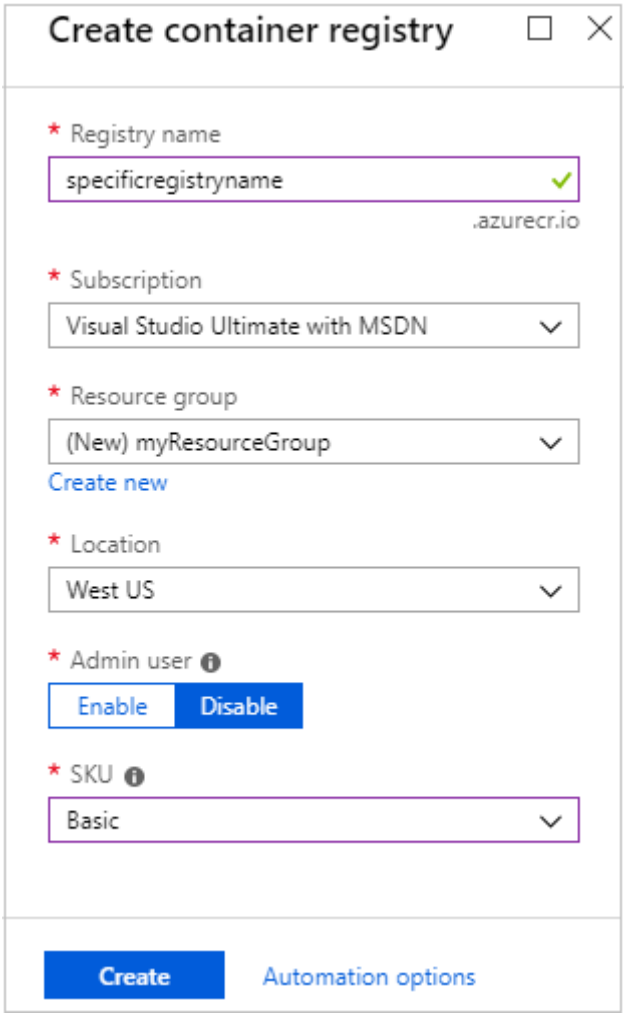


Figura 3-10. Crear un registro de contenedor

Una vez que haya creado el registro, deberá autenticarse con él para poder usarlo. Normalmente, iniciará sesión en el registro mediante el comando de la CLI de Azure:

Azure CLI

```
az acr login --name *registryname*
```

Una vez autenticado, puede usar comandos de Docker para insertar imágenes de contenedor en él. Sin embargo, para poder hacerlo, debe etiquetar la imagen con el nombre completo (URL) del servidor de inicio de sesión de ACR. Tendrá el formato *registryname.azurecr.io*.

Consola

```
docker tag mycontainer myregistry.azurecr.io/mycontainer:v1
```

Después de etiquetar la imagen, use el `docker push` comando para insertar la imagen en la instancia de ACR.

Consola

```
docker push myregistry.azurecr.io/mycontainer:v1
```

Después de insertar una imagen en el registro, se recomienda quitar la imagen del entorno de Docker local mediante este comando:

Consola

```
docker rmi myregistry.azurecr.io/mycontainer:v1
```

Como procedimiento recomendado, no debe insertar manualmente imágenes en un registro de contenedor. En su lugar, use una canalización de compilación definida en una herramienta como GitHub o Azure DevOps. Obtenga más información en el capítuloCloud-Native DevOps.

Tareas de ACR

ACR Tasks es un conjunto de características disponibles en Azure Container Registry. Amplía el ciclo interno de desarrollo a través de la creación y administración de imágenes de contenedor en la nube de Azure. En lugar de invocar un `docker build` y un `docker push` localmente en tu máquina de desarrollo, son gestionados automáticamente por las tareas de ACR en la nube.

El siguiente comando de la CLI de AZ compila una imagen de contenedor e lo inserta en ACR:

Azure CLI

```
# create a container registry
az acr create --resource-group myResourceGroup --name myContainerRegistry008 --sku
Basic

# build container image in ACR and push it into your container registry
```

```
az acr build --image sample/hello-world:v1  --registry myContainerRegistry008 --
file Dockerfile .
```

Como puede ver en el bloque de comandos anterior, no es necesario instalar Docker Desktop en la máquina de desarrollo. Además, puede configurar los desencadenadores de tareas de ACR para que recompilen imágenes de contenedores cuando se actualizan tanto el código fuente como las imágenes base.

Azure Kubernetes Service

En este capítulo se describe Azure Kubernetes Service (AKS). Hemos visto que es el orquestador de contenedores de hecho que administra aplicaciones nativas de nube contenedorizadas.

Una vez implementada una imagen en un registro, como ACR, puede configurar AKS para extraer e implementarla automáticamente. Con una canalización de CI/CD implementada, puede configurar una estrategia de versión controlada para minimizar el riesgo que conlleva implementar rápidamente actualizaciones. La nueva versión de la aplicación se configura inicialmente en producción sin tráfico que se enrute a ella. A continuación, el sistema enrutará un pequeño porcentaje de usuarios a la versión recién implementada. A medida que el equipo obtiene confianza en la nueva versión, puede implementar más instancias y retirar el antiguo. AKS admite fácilmente este estilo de implementación.

Al igual que con la mayoría de los recursos de Azure, puede crear un clúster de Azure Kubernetes Service mediante el portal, la línea de comandos o las herramientas de automatización como Helm o Terraform. Para empezar a trabajar con un nuevo clúster, debe proporcionar la siguiente información:

- Suscripción de Azure
- Grupo de recursos
- Nombre del clúster de Kubernetes
- Región
- Versión de Kubernetes
- Prefijo del nombre DNS
- Tamaño del nodo
- Recuento de nodos

Esta información es suficiente para empezar. Como parte del proceso de creación en Azure Portal, también puede configurar opciones para las siguientes características del clúster:

- Escala
- Autenticación

- Redes
- Monitorización
- Etiquetas

En este inicio rápido se explica cómo implementar un clúster de AKS mediante Azure Portal.

Azure Bridge to Kubernetes

Las aplicaciones nativas de la nube pueden crecer de forma grande y compleja, lo que requiere que se ejecuten recursos de proceso significativos. En estos escenarios, toda la aplicación no se puede hospedar en una máquina de desarrollo (especialmente un portátil). Azure Bridge to Kubernetes aborda la deficiencia. Permite a los desarrolladores trabajar con una versión local de su servicio mientras hospedan toda la aplicación en un clúster de desarrollo de AKS.

Cuando esté listo, los desarrolladores prueban sus cambios localmente mientras se ejecutan en la aplicación completa del clúster de AKS, sin replicar dependencias. En segundo plano, el puente combina el código de la máquina local con los servicios de AKS. Los desarrolladores pueden iterar rápidamente y depurar código directamente en Kubernetes usando Visual Studio o Visual Studio Code.

Gabe Monroy, ex vicepresidente de administración de productos en Microsoft, lo describe bien:

> Imagine que es un nuevo empleado que intenta corregir un error en una aplicación compleja de microservicios que consta de docenas de componentes, cada uno con su propia configuración y servicios de respaldo. Para empezar, debe configurar el entorno de desarrollo local para que pueda imitar la producción, incluida la configuración del IDE, la cadena de herramientas de creación, las dependencias de servicio en contenedor, un entorno local de Kubernetes, los simulacros para los servicios de respaldo, etc. Con todo el tiempo implicado en la configuración del entorno de desarrollo, corregir ese primer error podría tardar días. O bien, podría usar Bridge to Kubernetes y AKS.

Siguiente anterior

Escalado de contenedores y aplicaciones sin servidor

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Hay dos maneras de escalar una aplicación: vertical o horizontal. El primero hace referencia a agregar capacidad a un único recurso, mientras que este último hace referencia a agregar más recursos para aumentar la capacidad.

La solución sencilla: ampliar

La actualización de un servidor host existente con mayor rendimiento de CPU y memoria, así como un mejor rendimiento de E/S de disco y E/S de red, se conoce como *escalado vertical*. El escalado vertical de una aplicación nativa de la nube implica elegir recursos más capaces del proveedor de nube. Por ejemplo, puede crear un nuevo grupo de nodos con máquinas virtuales más grandes en el clúster de Kubernetes. A continuación, migre los servicios contenedorizados al nuevo grupo.

Para escalar las aplicaciones sin servidor, elija el plan premium de Functions o los tamaños de instancia premium de un plan dedicado de servicio de aplicaciones.

Escalamiento de aplicaciones nativas en la nube

Las aplicaciones nativas de la nube suelen experimentar grandes fluctuaciones en la demanda y requieren escala en un momento dado. Favorecen el escalado horizontal. El escalado horizontal se realiza mediante la adición de máquinas adicionales (denominadas nodos) o instancias de aplicación a un clúster existente. En Kubernetes, puede escalar manualmente ajustando las opciones de configuración de la aplicación (por ejemplo, escalando un grupo de nodos) o mediante el escalado automático.

Los clústeres de AKS pueden escalar automáticamente de una de estas dos maneras:

En primer lugar, Horizontal Pod Autoscaler supervisa la demanda de recursos y escala automáticamente las réplicas de POD para satisfacer dicha demanda. Cuando aumenta el tráfico, se aprovisionan automáticamente réplicas adicionales para escalar horizontalmente los servicios. Del mismo modo, cuando la demanda disminuye, se quitan para ajustar los servicios. Defina la métrica en la que se va a escalar, por ejemplo, el uso de cpu. También puede especificar el número mínimo y máximo de réplicas que se van a ejecutar. AKS supervisa esa métrica y escala en consecuencia.

A continuación, la característica Escalador automático de clústeres de AKS permite escalar automáticamente los nodos de proceso en un clúster de Kubernetes para satisfacer la demanda. Con ella, puede agregar automáticamente nuevas máquinas virtuales al conjunto de escalado de máquinas virtuales de Azure subyacente siempre que se requiera más capacidad de proceso. También quita los nodos cuando ya no son necesarios.

En la figura 3-11 se muestra la relación entre estos dos servicios de escalado.

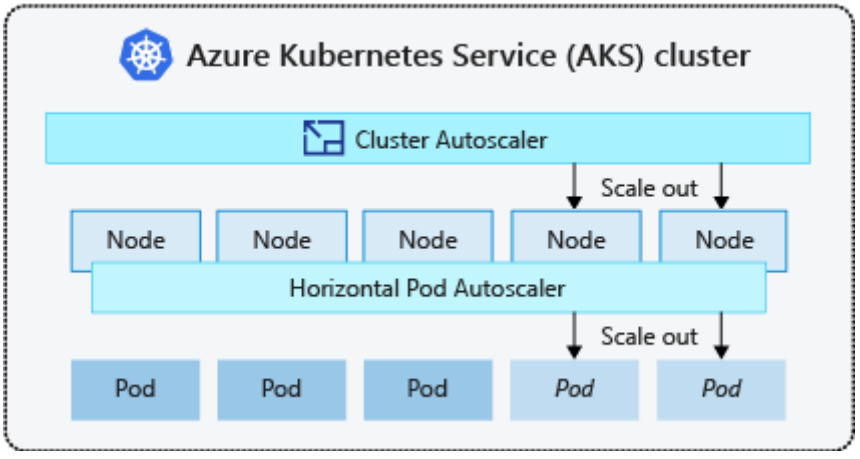


Figura 3-11. Escalado horizontal de un plan de App Service.

Trabajando juntos, garantizan un número óptimo de instancias de contenedor y nodos de cálculo para admitir la demanda fluctuante. El escalador automático horizontal de pods optimiza el número de pods necesarios. El escalador automático del clúster optimiza el número de nodos necesarios.

Escalado de las funciones de Azure

Azure Functions se escala horizontalmente de forma automática bajo demanda. Los recursos del servidor se asignan y quitan dinámicamente en función del número de eventos desencadenados. Solo se le cobra por los recursos de proceso consumidos cuando se ejecutan las funciones. La facturación se basa en el número de ejecuciones, el tiempo de ejecución y la memoria usadas.

Aunque el plan de consumo predeterminado proporciona una solución económica y escalable para la mayoría de las aplicaciones, la opción Premium permite a los desarrolladores flexibilidad para los requisitos personalizados de Azure Functions. La actualización al plan Premium proporciona control sobre los tamaños de instancia, las instancias previamente activadas (para evitar retrasos en el arranque en frío) y las máquinas virtuales dedicadas.

Siguiente anterior

Otras opciones de implementación de contenedores

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Además de Azure Kubernetes Service (AKS), también puede implementar contenedores en Azure App Service for Containers y Azure Container Instances.

¿Cuándo tiene sentido implementar en App Service for Containers?

Las aplicaciones de producción sencillas que no requieren orquestación son adecuadas para Azure App Service para contenedores.

Implementación en App Service para contenedores

Para realizar la implementación en Azure App Service para contenedores, necesitará una instancia de Azure Container Registry (ACR) y credenciales para acceder a ella. Inserte la imagen de contenedor en el repositorio de ACR para que Azure App Service pueda extraerla cuando sea necesario. Una vez completada, puede configurar la aplicación para la implementación continua. Al hacerlo, se implementarán automáticamente actualizaciones cada vez que cambie la imagen en ACR.

¿Cuándo tiene sentido implementar en Azure Container Instances?

Azure Container Instances (ACI) permite ejecutar contenedores de Docker en un entorno de nube administrado sin servidor, sin tener que configurar máquinas virtuales o clústeres. Es una excelente solución para cargas de trabajo de ejecución corta que se pueden ejecutar en un contenedor aislado. Considere ACI para servicios sencillos, escenarios de prueba, automatización de tareas y trabajos de compilación. ACI pone en marcha una instancia de contenedor, realiza la tarea y, luego, la detiene.

Implementación de una aplicación en Azure Container Instances

Para realizar la implementación en Azure Container Instances (ACI), necesita una instancia de Azure Container Registry (ACR) y credenciales para acceder a ella. Una vez que inserte la imagen de contenedor en el repositorio, estará disponible para extraerla en ACI. Puede trabajar con ACI mediante Azure Portal o la interfaz de la línea de comandos. ACR proporciona una estrecha integración con ACI. En la figura 3-12 se muestra cómo insertar una imagen de contenedor individual en ACR.

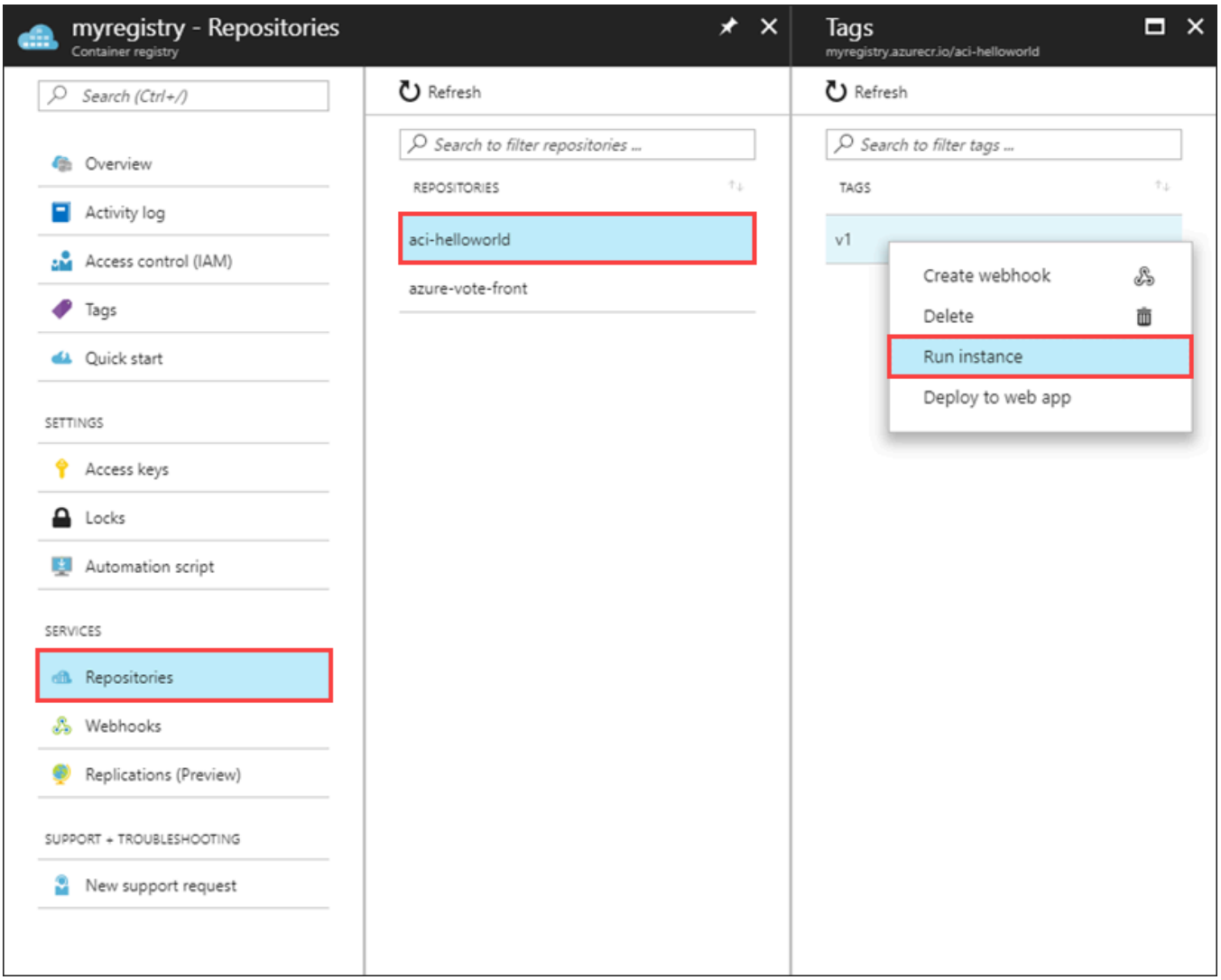


Figura 3-12. Instancia de ejecución de Azure Container Registry

La creación de una instancia en ACI se puede realizar rápidamente. Especifique el registro de imágenes, la información del grupo de recursos de Azure, la cantidad de memoria que se va a asignar y el puerto en el que se va a escuchar. En este inicio rápido se muestra cómo implementar una instancia de contenedor en ACI mediante Azure Portal.

Una vez completada la implementación, busque la dirección IP del contenedor recién implementada y conéctese con ella a través del puerto especificado.

Azure Container Instances ofrece la manera más rápida de ejecutar cargas de trabajo de contenedor sencillas en Azure. No es necesario configurar un servicio de aplicaciones, un orquestador o una máquina virtual. En escenarios en los que se requiere orquestación completa de contenedores, detección de servicios, escalado automático o actualizaciones coordinadas, se recomienda Azure Kubernetes Service (AKS).

Referencias

- ¿Qué es Kubernetes?

- Instalación de Kubernetes con Minikube
- MiniKube frente a Docker Desktop
- Visual Studio Tools para Docker
- Descripción del arranque en frío sin servidor
- Instancias de Azure Functions previamente activadas
- Creación de una función en Linux mediante una imagen personalizada
- Ejecución de Azure Functions en un contenedor de Docker
- Creación de una función en Linux mediante una imagen personalizada
- Escalado automático controlado por eventos de Azure Functions con Kubernetes
- Versión del valor controlado
- Azure Dev Spaces con VS Code
- Azure Dev Spaces con Visual Studio
- Grupos de varios nodos de AKS
- AKS Cluster Autoscaler
- Tutorial: Escalado de aplicaciones en AKS
- Escalado y hospedaje de Azure Functions
- Documentación de Azure Container Instances
- Implementación de Container Instance desde ACR

Siguiente anterior

Patrones de comunicación nativa en la nube

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico "Architecting Cloud Native .NET Applications for Azure" (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Al construir un sistema nativo de la nube, la comunicación se convierte en una decisión de diseño significativa. ¿Cómo se comunica una aplicación cliente de front-end con un microservicio de back-end? ¿Cómo se comunican los microservicios back-end entre sí? ¿Cuáles son los principios, patrones y procedimientos recomendados que se deben tener en cuenta al implementar la comunicación en aplicaciones nativas de la nube?

Consideraciones de comunicación

En una aplicación monolítica, la comunicación es sencilla. Los módulos de código se ejecutan juntos en el mismo espacio ejecutable (proceso) en un servidor. Este enfoque puede tener ventajas de rendimiento, ya que todo se ejecuta junto en memoria compartida, pero da como resultado código estrechamente acoplado que resulta difícil de mantener, evolucionar y escalar.

Los sistemas nativos de la nube implementan una arquitectura basada en microservicios con muchos microservicios pequeños e independientes. Cada microservicio se ejecuta en un proceso independiente y normalmente se ejecuta dentro de un contenedor que se implementa en un *clúster*.

Un clúster agrupa un grupo de máquinas virtuales para formar un entorno de alta disponibilidad. Se administran con una herramienta de orquestación, que es responsable de implementar y administrar los microservicios en contenedores. En la figura 4-1 se muestra un clúster de Kubernetes implementado en la nube de Azure con Azure Kubernetes Services totalmente administrado.

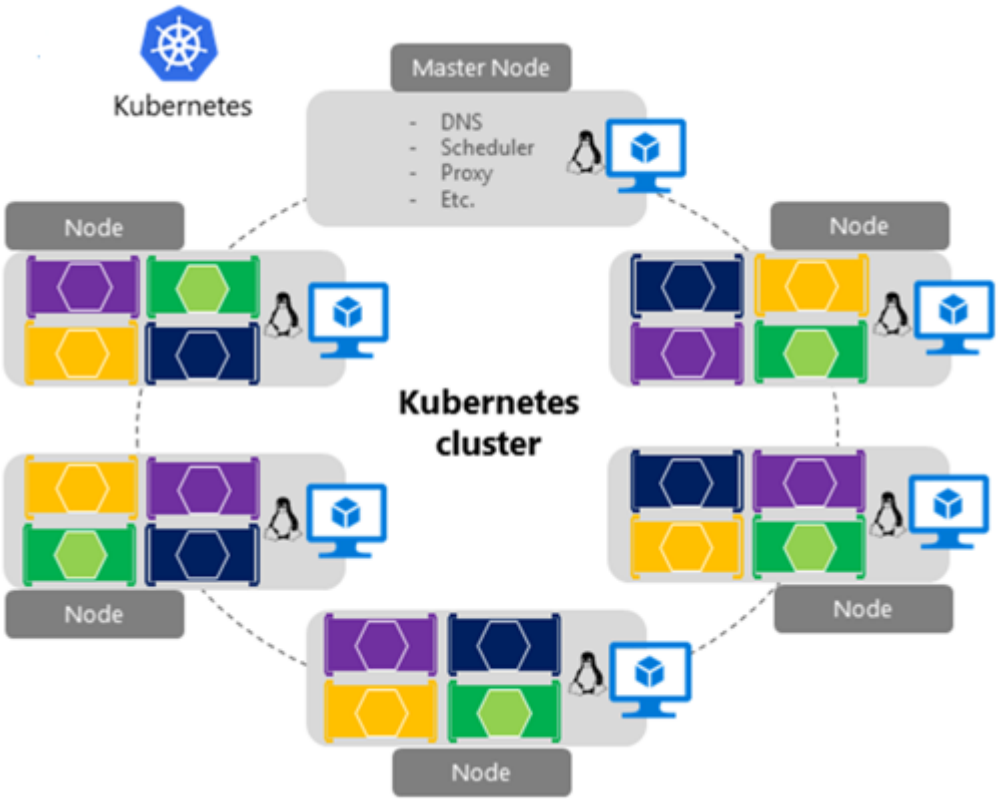


Figura 4-1. Un clúster de Kubernetes en Azure

En todo el clúster, los microservicios se comunican entre sí a través de las API y las tecnologías de mensajería.

Aunque proporcionan muchas ventajas, los microservicios no dan nada a cambio de nada. Las llamadas de método locales dentro de proceso entre componentes son reemplazadas por llamadas de red. Cada microservicio debe comunicarse a través de un protocolo de red, lo que agrega complejidad al sistema:

- la congestión de la red, la latencia y los errores transitorios son un problema constante.
- La resistencia (es decir, reintentar solicitudes con error) es esencial.
- Algunas llamadas deben ser idempotentes para mantener un estado coherente.
- Cada microservicio debe autenticar y autorizar llamadas.
- Cada mensaje debe serializarse y después deserializarse, lo que puede ser costoso.

- El cifrado y descifrado de mensajes se vuelve importante.

El libro ".NET Microservices: Architecture for Containerized .NET Applications" (Microservicios de .NET: Arquitectura para aplicaciones .NET contenedorizadas), disponible de forma gratuita desde Microsoft, proporciona una cobertura detallada de los patrones de comunicación para las aplicaciones de microservicios. En este capítulo se proporciona información general de alto nivel de estos patrones junto con las opciones de implementación disponibles en la nube de Azure.

En este capítulo abordaremos primero la comunicación entre las aplicaciones de front-end y los microservicios de back-end. A continuación veremos los microservicios de back-end que se comunican entre sí. Exploraremos la tecnología de comunicación up y gRPC. Por último, veremos nuevos patrones de comunicación innovadores mediante la tecnología de malla de servicio. También veremos cómo la nube de Azure proporciona diferentes tipos de *servicios de respaldo* para admitir la comunicación nativa de la nube.

Anterior Siguiente

Comunicación de cliente front-end

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico “Architecting Cloud Native .NET Applications for Azure” (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

En un sistema nativo de nube, los clientes front-end (aplicaciones móviles, web y de escritorio) requieren un canal de comunicación para interactuar con microservicios back-end independientes.

¿Cuáles son las opciones?

Para simplificar las cosas, un cliente front-end podría *comunicarse directamente* con los microservicios back-end, como se muestra en la figura 4-2.

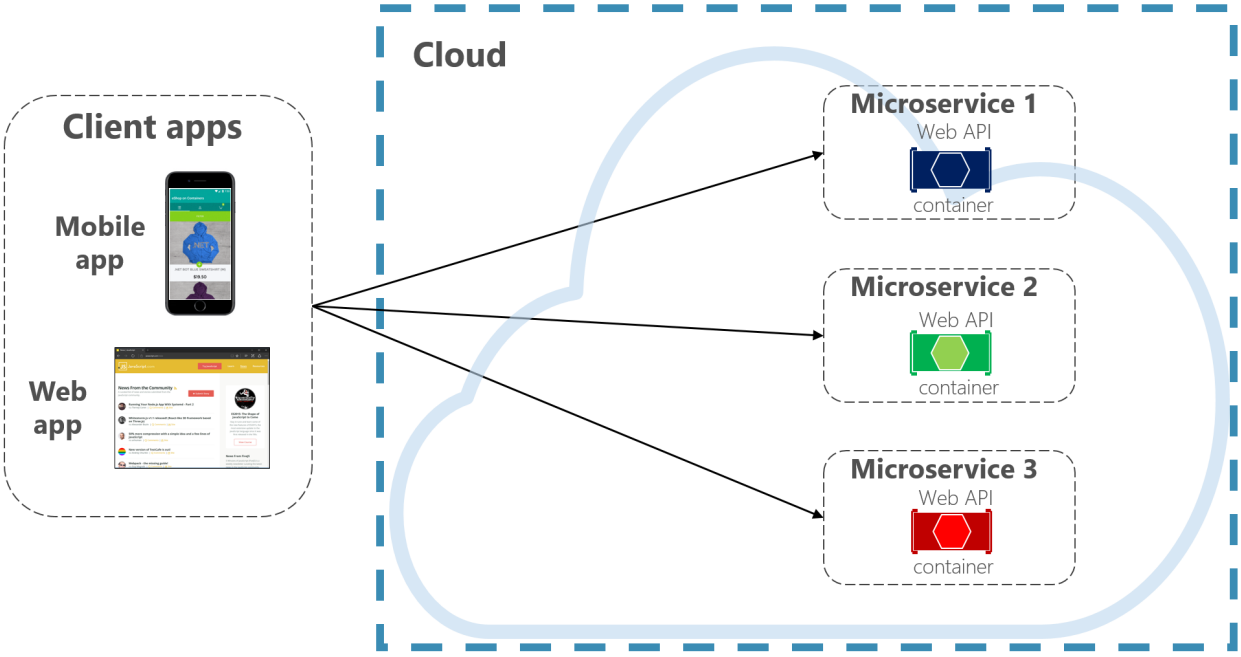


Figura 4-2. Comunicación directa de cliente a servicio

Con este enfoque, cada microservicio tiene un punto de conexión público al que pueden acceder los clientes front-end. En un entorno de producción, colocaría un equilibrador de carga delante de los microservicios, enrutando el tráfico proporcionalmente.

Aunque es fácil de implementar, la comunicación de cliente directa solo sería aceptable para aplicaciones de microservicio simples. Este patrón acopla estrechamente los clientes front-end a los servicios back-end principales, lo que abre la puerta a muchos problemas, entre los que se incluyen:

- Susceptibilidad del cliente a la refactorización del servicio back-end.
- Una superficie de ataque más amplia, al exponerse directamente los servicios back-end principales.
- Duplicación de problemas transversales entre cada microservicio.
- Código de cliente demasiado complejo: los clientes deben realizar un seguimiento de varios puntos de conexión y manejar los errores de una manera resistente.

En su lugar, un patrón de diseño en la nube ampliamente aceptado es implementar un Servicio de puerta de enlace de API entre las aplicaciones front-end y los servicios back-end. El patrón se muestra en la figura 4-3.

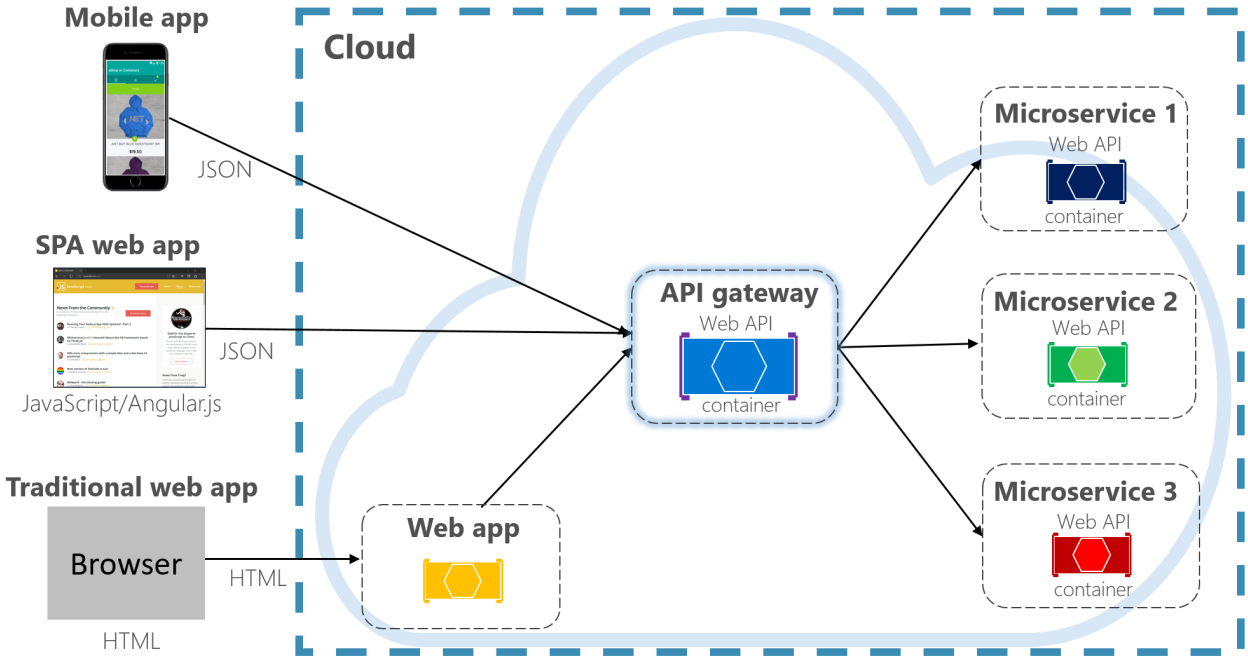


Figura 4-3. Patrón de puerta de enlace de API

En la ilustración anterior, observe cómo el servicio de puerta de enlace de API abstrae los microservicios principales de back-end. Implementado como API web, actúa como *proxy inverso* y enruta el tráfico entrante a los microservicios internos.

La puerta de enlace aísla al cliente de la creación de particiones y refactorización del servicio interno. Si cambia un servicio back-end, puede acomodarlo en la puerta de enlace sin dar problemas al cliente. También es la primera línea de defensa para cuestiones transversales, como la identidad, el almacenamiento en caché, la resistencia, la medición y la limitación. Muchos de estas cuestiones transversales se pueden descargar de los servicios principales back-end a la puerta de enlace, lo que simplifica los servicios back-end.

Debe tener cuidado para que la puerta de enlace de API sea sencilla y rápida. Normalmente, la lógica de negocio se mantiene fuera de la puerta de enlace. Una puerta de enlace compleja corre el riesgo de convertirse en un cuello de botella y, finalmente, en un monolito. Los sistemas más grandes suelen exponer varias puertas de enlace de API segmentadas por tipo de cliente (móvil, web, escritorio) o funcionalidad back-end. El patrón Back-end para front-end proporciona la dirección para implementar varias puertas de enlace. El patrón se muestra en la figura 4-4.

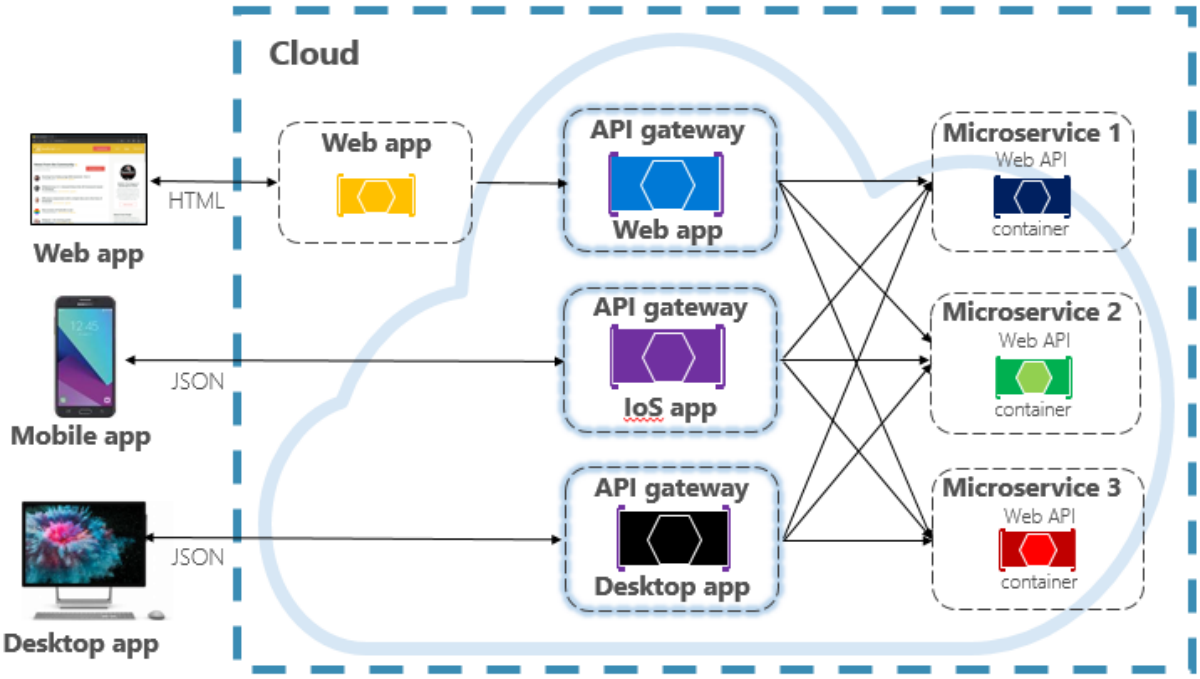


Figura 4-4. Patrón Back-end para front-end

Observe en la ilustración anterior cómo se envía el tráfico entrante a una puerta de enlace de API específica, en función del tipo de cliente: aplicación web, móvil o de escritorio. Este enfoque tiene sentido, ya que las funcionalidades de cada dispositivo difieren significativamente en función de las limitaciones de factor de forma, rendimiento y visualización. Normalmente, las aplicaciones móviles exponen menos funcionalidad que un explorador web o aplicaciones de escritorio. Cada puerta de enlace se puede optimizar para que coincida con las capacidades y funcionalidades del dispositivo correspondiente.

Puertas de enlace sencillas

Para empezar, puede crear su propio servicio de puerta de enlace de API. Una búsqueda rápida de GitHub proporcionará muchos ejemplos.

En el caso de aplicaciones sencillas nativas de nube de .NET, puede considerar la puerta de enlace de Ocelot. De código abierto y creado para microservicios de .NET, es ligera, rápida y escalable. Al igual que cualquier puerta de enlace de API, su funcionalidad principal es reenviar las solicitudes HTTP entrantes a los servicios de nivel inferior. Además, admite una amplia variedad de funcionalidades que se pueden configurar en una canalización de middleware de .NET.

YARP (Otro Proxy Inverso) es otro proxy inverso de código abierto dirigido por un grupo de equipos de productos de Microsoft. Descargable como un paquete NuGet,

YARP se conecta al marco ASP.NET como middleware y es muy personalizable. Encontrará YARP bien documentado con varios ejemplos de uso.

En el caso de las aplicaciones nativas de nube empresariales, hay varios servicios administrados de Azure que pueden ayudar a empezar a trabajar.

Azure Application Gateway

En el caso de los requisitos de puerta de enlace simples, puede considerar Azure Application Gateway. Disponible como un servicio PaaS de Azure, incluye características básicas de puerta de enlace, como el enrutamiento de direcciones URL, la terminación SSL y un Web Application Firewall. El servicio admite funcionalidades de equilibrio de carga de capa 7. Con capa 7 puede enrutar las solicitudes en función del contenido real de un mensaje HTTP, no solo de los paquetes de red TCP de bajo nivel.

En este libro evangelizamos sobre el hospedaje de sistemas nativos de nube en Kubernetes. Kubernetes, un orquestador de contenedores, automatiza la implementación, el escalado y las cuestiones operativas de las cargas de trabajo contenedorizadas. Azure Application Gateway se puede configurar como puerta de enlace de API para un clúster de Azure Kubernetes Service.

El Controlador de entrada de Application Gateway permite que Azure Application Gateway funcione directamente con Azure Kubernetes Service. En la figura 4.5 se muestra la arquitectura.

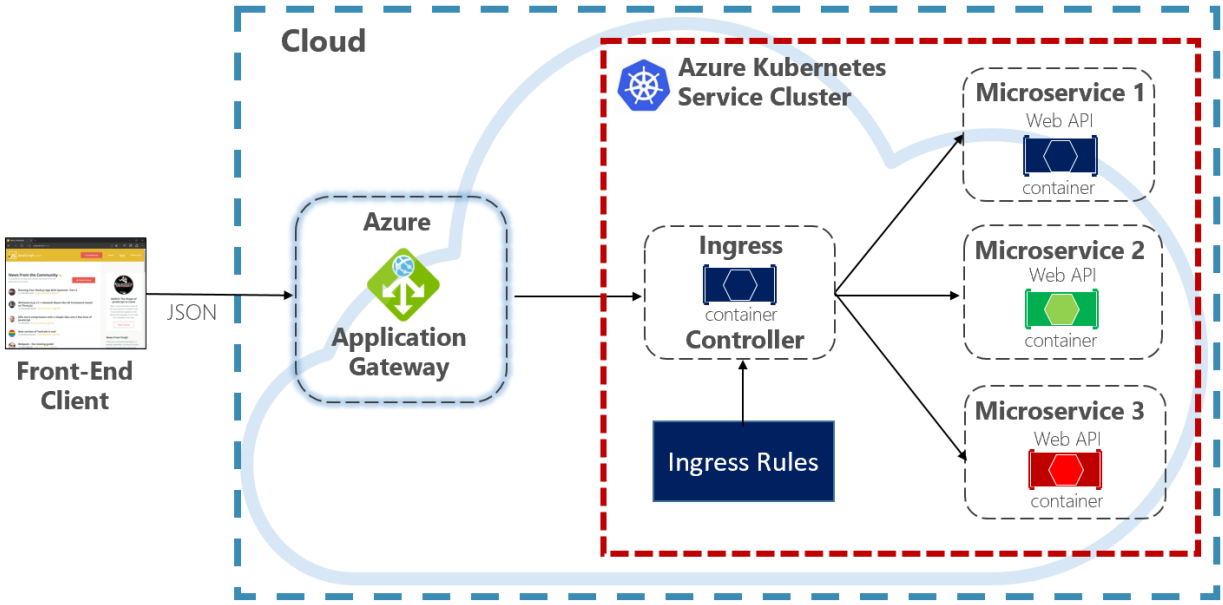


Figura 4-5. Controlador de entrada de Application Gateway

Kubernetes incluye una característica integrada que admite el equilibrio de carga HTTP (nivel 7), denominada Ingress. Ingress define un conjunto de reglas para la forma en

que las instancias de microservicio dentro de AKS se pueden exponer al mundo exterior. En la imagen anterior, el controlador de entrada interpreta las reglas de entrada configuradas para el clúster y configura automáticamente la Azure Application Gateway. En función de esas reglas, la Application Gateway enruta el tráfico a los microservicios que se ejecutan dentro de AKS. El controlador de entrada escucha los cambios en las reglas de entrada y realiza los cambios adecuados en la Azure Application Gateway.

Azure API Management

Para sistemas nativos de nube de gran escala a moderados, puede tener en cuenta Azure API Management. Se trata de un servicio basado en la nube que no solo resuelve las necesidades de la puerta de enlace de API, sino que proporciona una experiencia completa de desarrollador y administrador. API Management se muestra en la figura 4-6.

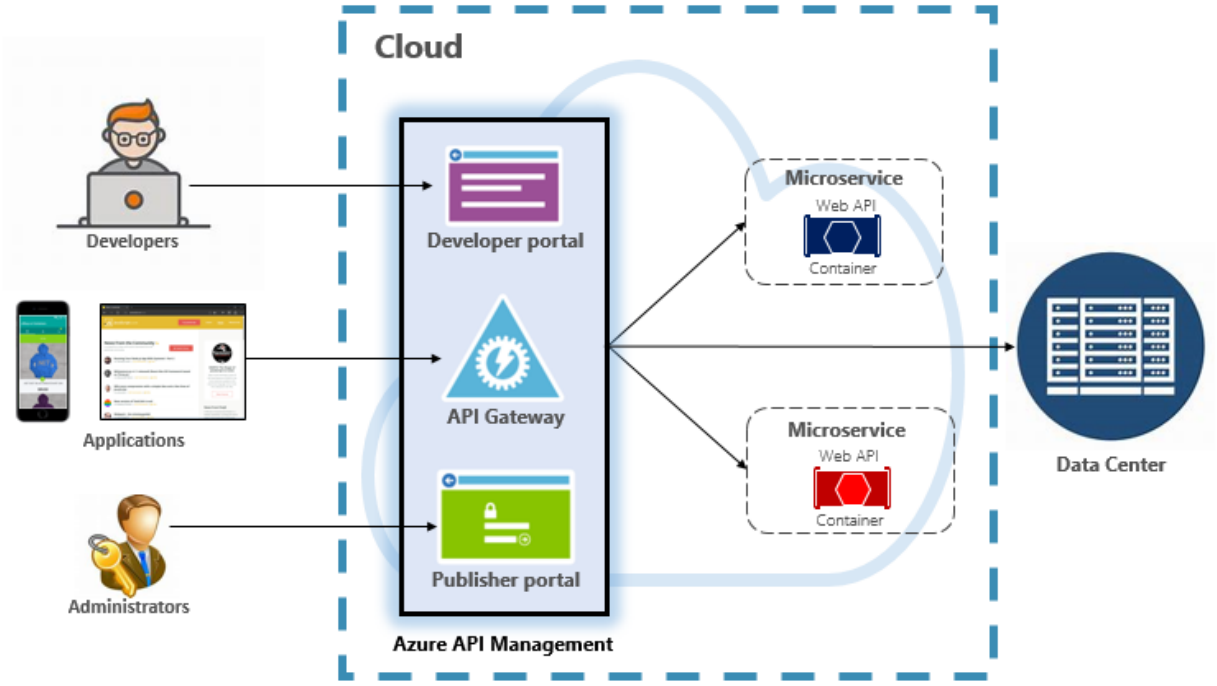


Figura 4-6. Azure API Management

Para empezar, API Management expone un servidor de puerta de enlace que permite el acceso controlado a los servicios back-end en función de reglas y directivas configurables. Estos servicios pueden estar en la nube de Azure, en el centro de datos locales o en otras nubes públicas. Las claves de API y los tokens JWT determinan quién puede hacer qué. Todo el tráfico se registra con fines analíticos.

Para desarrolladores, API Management ofrece un portal para desarrolladores que proporciona acceso a servicios, documentación y código de ejemplo para invocarlos. Los desarrolladores pueden usar Swagger/Open API para inspeccionar los puntos de

conexión de servicio y analizar su uso. El servicio funciona entre las principales plataformas de desarrollo: .NET, Java, Golang, etc.

El portal del publicador expone un panel de administración donde los administradores exponen las API y administran su comportamiento. Se puede conceder acceso al servicio, supervisar el estado del servicio y recopilar la telemetría del servicio. Los administradores aplican *directivas* a cada punto de conexión para afectar al comportamiento. Las directivas son instrucciones precompiladas que se ejecutan secuencialmente para cada llamada de servicio. Las directivas se configuran para una llamada entrante, una llamada saliente o se invocan en caso de error. Las directivas se pueden aplicar en distintos ámbitos de servicio, así como para habilitar la ordenación determinista al combinar directivas. El producto se distribuye con un gran número de directivas precompiladas.

Estos son ejemplos de cómo pueden afectar las directivas al comportamiento de los servicios nativos de nube:

- Restringir el acceso al servicio.
- Exigir autenticación.
- Limitar las llamadas desde un único origen, si es necesario.
- Activación del almacenamiento en caché.
- Bloquear llamadas desde direcciones IP específicas.
- Controlar el flujo del servicio.
- Convertir solicitudes de SOAP a REST o entre diferentes formatos de datos, como de XML a JSON.

Azure API Management puede exponer servicios back-end hospedados en cualquier lugar, en la nube o en el centro de datos. En el caso de los servicios heredados que puede exponer en los sistemas nativos de la nube, admite las API de REST y de SOAP. Incluso otros servicios de Azure se pueden exponer mediante API Management. Podría colocar una API administrada sobre un servicio de respaldo de Azure, como Azure Service Bus o Azure Logic Apps. Azure API Management no incluye compatibilidad integrada con el equilibrio de carga y debe usarse junto con un servicio de equilibrio de carga.

Azure API Management está disponible en cuatro niveles de servicio diferentes:

- Desarrollador
- Básico
- Estándar
- Premium

El nivel Desarrollador está pensado para cargas de trabajo y evaluación que no son de producción. Los demás niveles ofrecen cada vez más potencia, características y contratos de nivel de servicio (SLA) superiores. El nivel Premium proporciona compatibilidad con Azure Virtual Network y varias regiones. Todos los niveles tienen un precio fijo por hora.

La nube de Azure también ofrece un nivel sin servidor para Azure API Management. Conocido como *plan de tarifa de consumo*, el servicio es una variante de API Management diseñada en torno al modelo de informática sin servidor. A diferencia de los planes de tarifa "asignados previamente" mostrados anteriormente, el nivel de consumo proporciona precios instantáneos de aprovisionamiento y pago por acción.

Habilita las características de puerta de enlace de API para los siguientes casos de uso:

- Microservicios implementados mediante tecnologías sin servidor, como Azure Functions y Azure Logic Apps.
- Recursos del servicio de respaldo de Azure, como colas de Service Bus y temas, Azure Storage, etc.
- Microservicios en los que el tráfico tiene picos grandes ocasionales, pero sigue siendo bajo la mayor parte del tiempo.

El nivel de consumo usa el mismo servicio subyacente de componentes de API Management, pero emplea una arquitectura completamente diferente basada en recursos asignados dinámicamente. Se alinea perfectamente con el modelo de informática sin servidor:

- No hay que administrar ninguna infraestructura.
- No hay capacidad inactiva.
- Alta disponibilidad.
- Escalado automático
- El coste se basa en el uso real.

El nuevo nivel de consumo es una excelente opción para los sistemas nativos de nube que exponen recursos sin servidor como API.

Comunicación en tiempo real

La comunicación en tiempo real o de inserción es otra opción para las aplicaciones front-end que se comunican con sistemas back-end nativos de nube a través de HTTP. Las aplicaciones, como los teletipos financieros, la educación en línea, los juegos y las actualizaciones de progreso del trabajo, requieren respuestas instantáneas en tiempo real del back-end. Con la comunicación HTTP normal, no hay ninguna manera de que el

cliente sepa cuándo hay nuevos datos disponibles. El cliente debe *sondear* o enviar continuamente solicitudes al servidor. Con la comunicación *en tiempo real*, el servidor puede insertar nuevos datos al cliente en cualquier momento.

Los sistemas en tiempo real suelen caracterizarse por flujos de datos de alta frecuencia y un gran número de conexiones de cliente simultáneas. La implementación manual de la conectividad en tiempo real puede convertirse rápidamente en compleja, al requerir una infraestructura no despreciable para garantizar la escalabilidad y una mensajería confiable a los clientes conectados. Podría encontrarse administrando una instancia de Azure Redis Cache y un conjunto de equilibradores de carga configurados con sesiones permanentes para la afinidad de cliente.

Azure SignalR Service es un servicio de Azure totalmente administrado que simplifica la comunicación en tiempo real para las aplicaciones nativas de la nube. Los detalles de implementación técnica, como el aprovisionamiento de capacidad, el escalado y las conexiones persistentes, se abstrae. Se administran por usted con un contrato de nivel de servicio del 99,9 %. Usted se centra en las características de la aplicación, no en la canalización de la infraestructura.

Una vez habilitado, un servicio HTTP basado en la nube puede insertar actualizaciones de contenido directamente a los clientes conectados, incluidas las aplicaciones de explorador, móviles y de escritorio. Los clientes se actualizan sin necesidad de sondear el servidor. Azure SignalR abstrae las tecnologías de transporte que crean conectividad en tiempo real, incluidos WebSockets, Server-Side Events y Long Polling. Los desarrolladores se centran en enviar mensajes a todos los subconjuntos o a subconjuntos específicos de clientes conectados.

En la figura 4-7 se muestra un conjunto de clientes HTTP que se conectan a una aplicación nativa de nube con Azure SignalR habilitado.

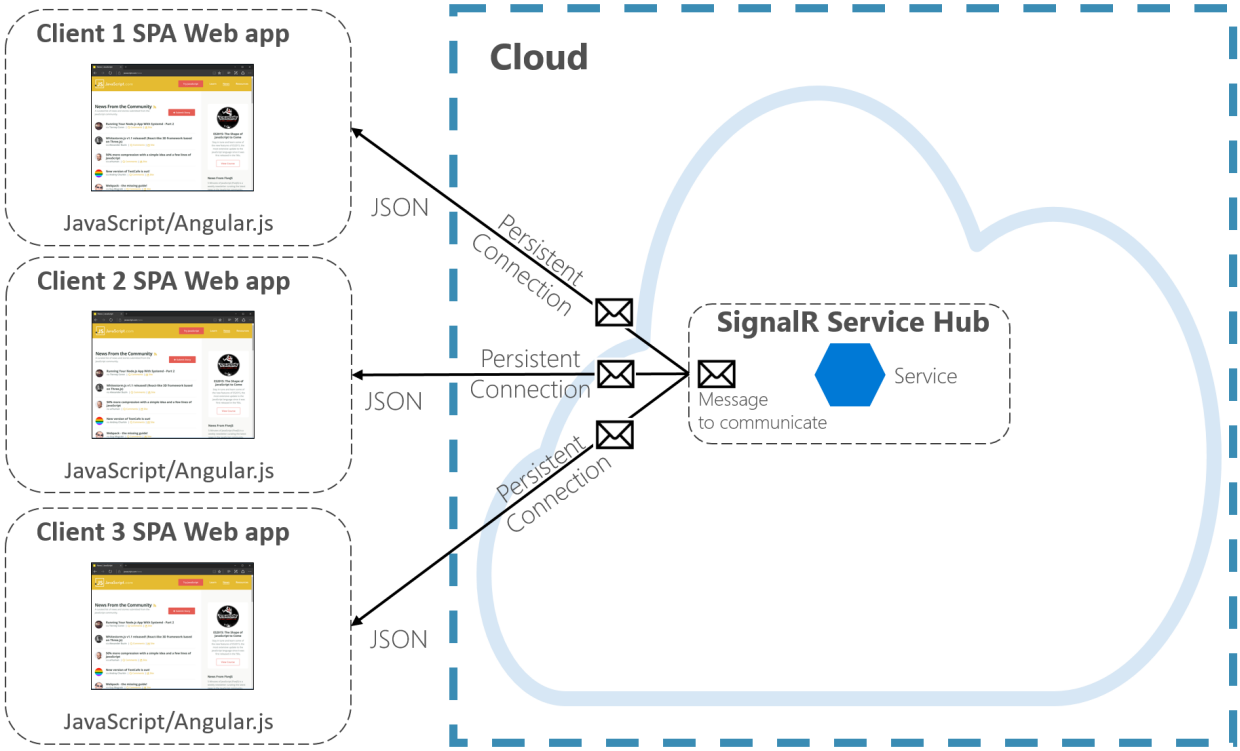


Figura 4-7. Azure SignalR

Otra ventaja de Azure SignalR Service aparece con la implementación de servicios nativos de nube sin servidor. Quizás el código se ejecute bajo demanda con desencadenadores de Azure Functions. Este escenario puede ser complicado porque el código no mantiene conexiones largas con los clientes. Azure SignalR Service puede encargarse de esta situación, ya que el servicio ya administra las conexiones por usted.

Azure SignalR Service se integra estrechamente con otros servicios de Azure, como Azure SQL Database, Service Bus o Redis Cache, lo que abre muchas posibilidades para las aplicaciones nativas de nube.

Anterior Siguiente

Comunicación entre servicios

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico “Architecting Cloud Native .NET Applications for Azure” (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Tras el cliente front-end, pasemos ahora a ver cómo se comunican los microservicios back-end entre sí.

Al crear una aplicación nativa de nube, nos interesa que sea consciente de cómo se comunican los servicios back-end entre sí. Lo ideal es que cuanta menos comunicación haya entre servicios, mejor. Sin embargo, no siempre es posible evitarlo, ya que los servicios back-end suelen depender los unos de otros para realizar operaciones.

Hay varios enfoques ampliamente aceptados para implementar la comunicación entre servicios. A menudo, el *tipo de interacción de la comunicación* será lo que determine cuál es el mejor enfoque.

Veamos los siguientes tipos de interacción:

- *Consulta*: un microservicio que realiza una llamada requiere una respuesta del microservicio al que ha llamado (por ejemplo, "Oye, pásame la información de comprador de un identificador de cliente determinado").
- *Comando*: un microservicio que realiza una llamada necesita que otro microservicio realice una acción, pero no requiere una respuesta (por ejemplo, "Oye, envía este pedido").

- *Evento*: un microservicio (el publicador) genera un evento que indica que un estado ha cambiado o que se ha producido una acción. Otros microservicios (los suscriptores), que están interesados, pueden reaccionar al evento como corresponda. El publicador y los suscriptores no tienen conocimiento el uno de los otros.

Normalmente, los sistemas de microservicios usan una combinación de estos tipos de interacción al realizar operaciones que requieren la interacción entre servicios. Echemos un vistazo a cada uno de ellos y cómo implementarlos.

Consultas

Muchas veces, un microservicio podría necesitar *consultar* otro, lo que requiere una respuesta inmediata para completar una operación. Un microservicio de cesta de la compra puede necesitar información del producto y un precio para agregar un artículo a la cesta. Hay muchos enfoques para implementar operaciones de consulta.

Mensajería de solicitud-respuesta

Una opción para implementar este escenario es que el microservicio back-end que realiza la llamada realice solicitudes HTTP directas a los microservicios que necesita consultar (como se muestra en la figura 4-8).

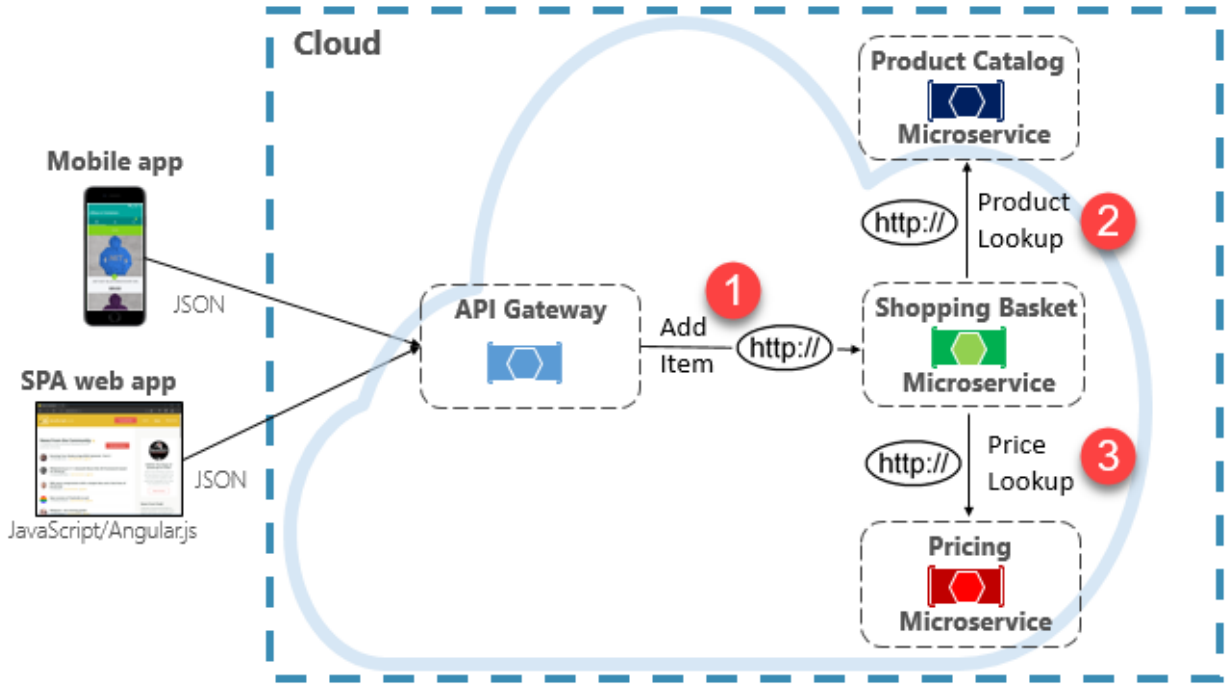


Figura 4-8. Comunicación HTTP directa

Aunque las llamadas HTTP directas entre microservicios son relativamente sencillas de implementar, conviene reducir esta práctica al mínimo. Para empezar, estas llamadas

siempre son *sincrónicas* y bloquearán la operación hasta que se obtenga un resultado o se agote el tiempo de espera de la solicitud. Lo que antes eran servicios autocontenidos e independientes, capaces de evolucionar de forma independiente e implementar frecuentemente, ahora son servicios acoplados entre sí. A medida que aumenta el acoplamiento entre microservicios, sus ventajas arquitectónicas van disminuyendo.

La ejecución de una solicitud poco frecuente que realiza una única llamada HTTP directa a otro microservicio puede ser aceptable en algunos sistemas. Sin embargo, las llamadas de gran volumen que invocan llamadas HTTP directas a varios microservicios no son aconsejables. Pueden aumentar la latencia y afectar negativamente al rendimiento, la escalabilidad y la disponibilidad del sistema. Peor aún, una serie prolongada de comunicación HTTP directa puede desembocar en una concatenación profunda y compleja de llamadas a microservicios sincrónicos (como se muestra en la figura 4-9):

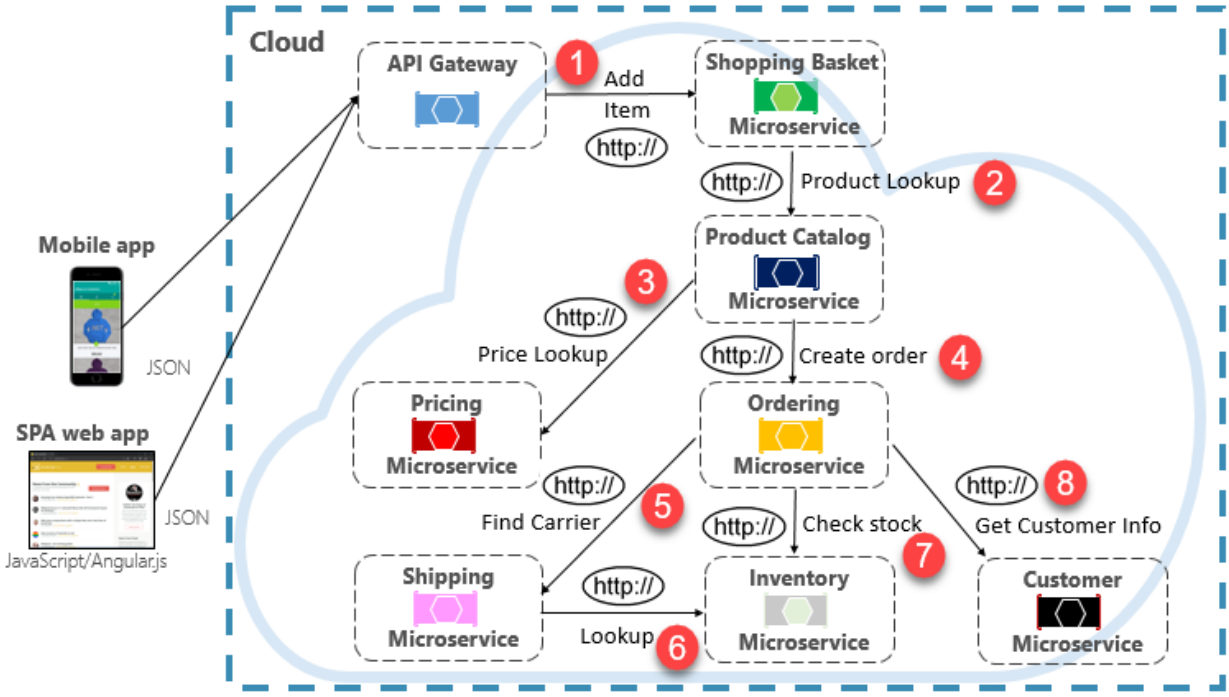


Figura 4-9. Concatenación de consultas HTTP

No resulta difícil imaginar el riesgo en el diseño que se trasluce en la imagen anterior. ¿Qué ocurre si se produce un error en el paso 3? ¿O en el paso 8? ¿Cómo nos recuperamos? ¿Qué ocurre si el paso 6 es lento porque el servicio subyacente está ocupado? ¿Cómo continuamos? Aun cuando todo funcione correctamente, pensemos en la latencia en que incurriría esta llamada, que sería la suma de la latencia de cada paso.

El enorme grado de acoplamiento de la imagen anterior sugiere que los servicios no se han modelado de forma óptima. Sería necesario que el equipo revisara su diseño.

Patrón Materialized View

Una opción popular para acabar con el acoplamiento de microservicios es el patrón de vista materializada. Con este patrón, un microservicio almacena su propia copia local y desnormalizada de los datos que pertenecen a otros servicios. En lugar de que el microservicio de cesta de la compra consulte los microservicios de catálogo de productos y precios, mantiene su propia copia local de esos datos. Este patrón elimina el acoplamiento innecesario y mejora el tiempo de confiabilidad y respuesta. Toda la operación se ejecuta dentro de un mismo proceso. Exploraremos este patrón y otras cuestiones relativas a los datos en el capítulo 5.

Patrón de agregador de servicio

Otra opción para acabar con el acoplamiento de microservicios es el microservicio de agregador, que se muestra de color púrpura en la figura 4-10.

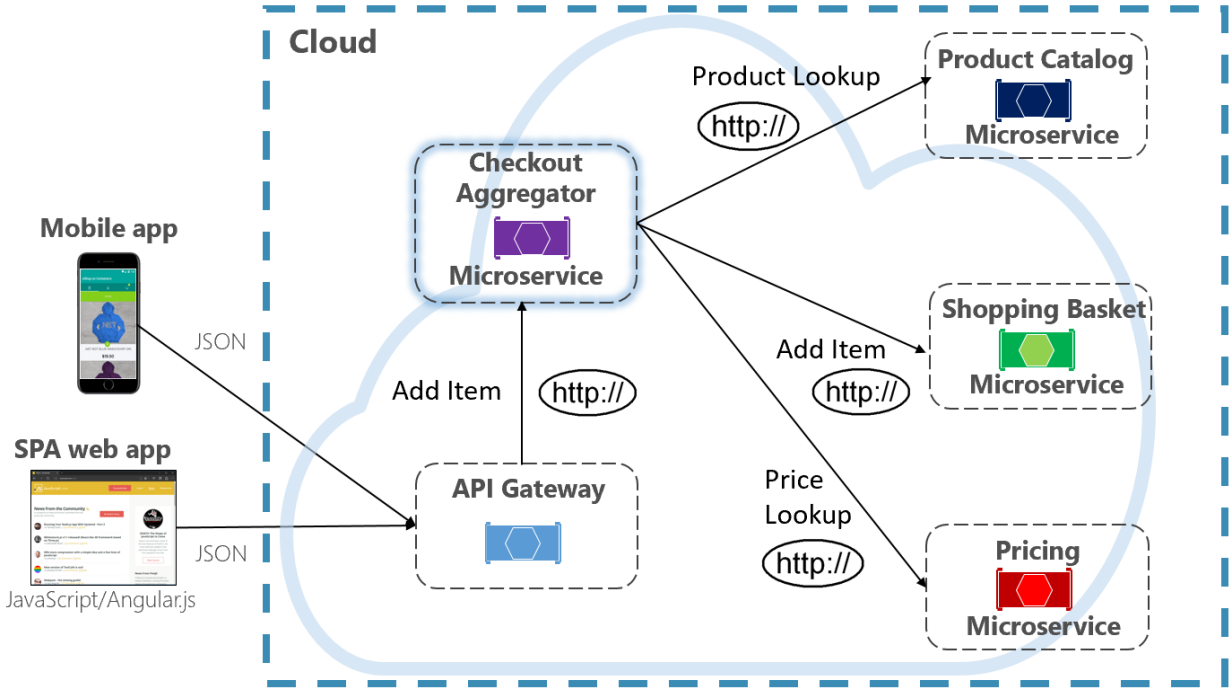


Figura 4-10. Microservicio de agregador

Este patrón aísla una operación que realiza llamadas a varios microservicios back-end, centralizando su lógica en un microservicio especializado. El microservicio de agregador de finalización de compra de la figura anterior (en color púrpura) organiza el flujo de trabajo para la operación de finalización de compra. Incluye llamadas a varios microservicios back-end en un orden secuenciado. Los datos del flujo de trabajo se agregan y se devuelven al autor de la llamada. Aunque sigue implementando llamadas HTTP directas, el microservicio de agregador reduce las dependencias directas entre los microservicios back-end.

Patrón de solicitud/respuesta

Otro enfoque para desacoplar mensajes HTTP sincrónicos es un patrón de solicitud/respuesta, que usa la comunicación de puesta en cola. La comunicación mediante una cola es siempre un canal unidireccional, con un productor que envía el mensaje y un consumidor que lo recibe. Con este patrón, se implementan una cola de solicitudes y una cola de respuestas (como se muestra en la figura 4-11).

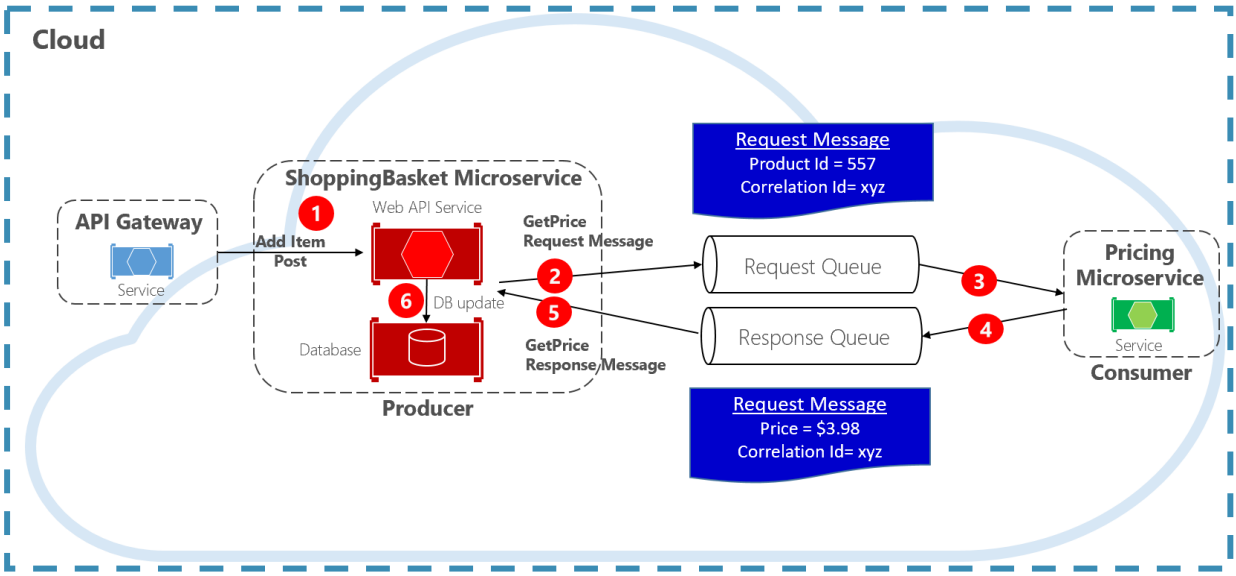


Figura 4-11. Patrón de solicitud/respuesta

Aquí, el productor del mensaje crea un mensaje basado en consultas que contiene un identificador de correlación único, y lo coloca en una cola de solicitudes. El servicio que consume quita el mensaje de la cola, lo procesa y coloca la respuesta en la cola de respuestas con el mismo identificador de correlación. El servicio productor quita el mensaje de la cola, lo coteja con el identificador de correlación y sigue procesando. Trataremos las colas en detalle en la sección siguiente.

Comandos

Otro tipo de interacción de comunicación es un *comando*. Un microservicio puede necesitar que otro microservicio realice una acción. El microservicio de pedidos puede necesitar que el microservicio de envíos cree el envío de un pedido aprobado. En la figura 4-12, un microservicio (el productor) envía un mensaje a otro microservicio (el consumidor), y le ordena que haga algo.

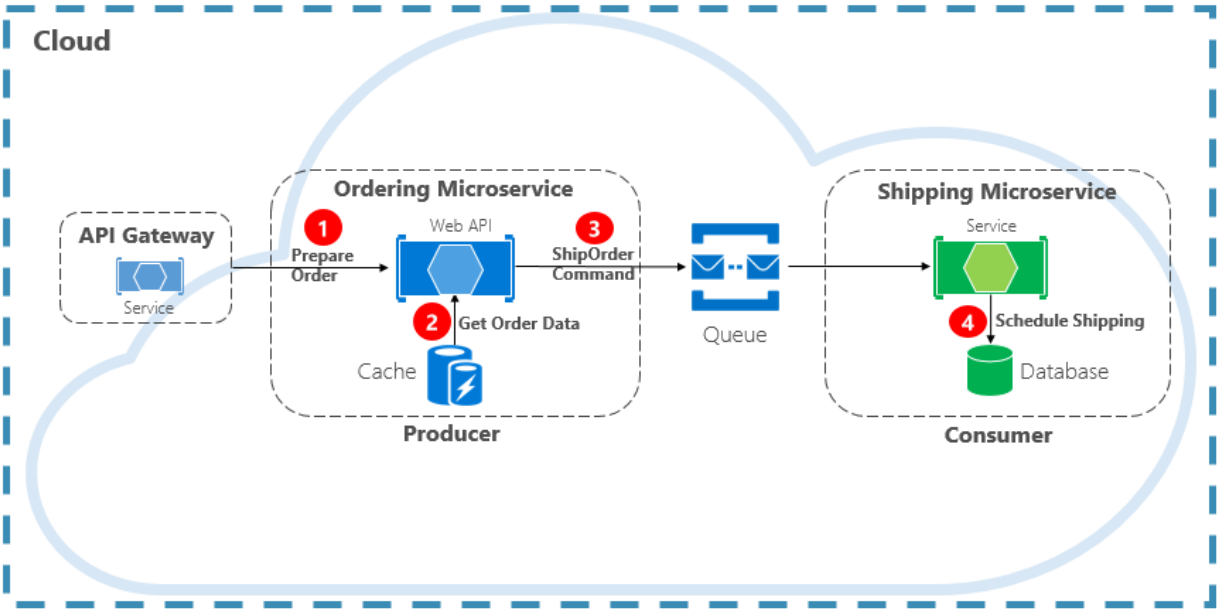


Figura 4-12. Interacción de comandos con una cola

La mayoría de las veces, el productor no requiere una respuesta y puede *descartar el mensaje de manera autónoma*. En caso de que sí se necesite una respuesta, el consumidor devuelve un mensaje independiente al productor en otro canal. La mejor forma de enviar un mensaje de comando es hacerlo de forma asincrónica con una cola de mensajes compatible con un agente de mensajes ligero. En el diagrama anterior, observe cómo una cola separa y desacopla ambos servicios.

Una cola de mensajes es una construcción intermedia a través de la cual un productor y consumidor pasan un mensaje. Las colas implementan un patrón de mensajería asincrónica de punto a punto. El productor sabe dónde se debe enviar un comando y lo enruta correctamente. La cola garantiza que un mensaje se procesa exactamente mediante una de las instancias de consumidor que leen el canal. En este escenario, tanto el servicio de productor como el de consumidor pueden escalar horizontalmente sin verse afectados entre sí. Además, las tecnologías pueden ser distintas en cada lado, lo que significa que podría haber un microservicio de Java que llame a un microservicio de Golang.

En el capítulo 1 vimos los *servicios de respaldo*. Los servicios de respaldo son recursos auxiliares de los que dependen los sistemas nativos de nube. Las colas de mensajes son servicios de respaldo. La nube de Azure admite dos tipos de colas de mensajes que los sistemas nativos de nube pueden consumir para implementar la mensajería de comandos: colas de Azure Storage y colas de Azure Service Bus.

Colas de Azure Storage

Las colas de Azure Storage ofrecen una infraestructura de colas sencilla que es rápida y asequible y que está respaldada por las cuentas de Azure Storage.

Las colas de Azure Storage incluyen un mecanismo de puesta en cola basado en REST con una mensajería confiable y persistente. Proporcionan un conjunto de características mínimo, pero son económicas y almacenan millones de mensajes. Su capacidad llega a los 500 TB. Un solo mensaje puede tener un tamaño máximo de 64 KB.

Se puede acceder a los mensajes desde cualquier lugar del mundo a través de llamadas autenticadas mediante HTTP o HTTPS. Las colas de Azure Storage pueden escalar horizontalmente a un gran número de clientes simultáneos para controlar los picos de tráfico.

Dicho esto, este servicio tiene sus limitaciones:

- El orden de los mensajes no está garantizado.
- Un mensaje se conserva solo siete días antes de que se quite automáticamente.
- No existe compatibilidad con la administración de estados, la detección de duplicados o las transacciones.

En la figura 4-13 se muestra la jerarquía de colas de Azure Storage.

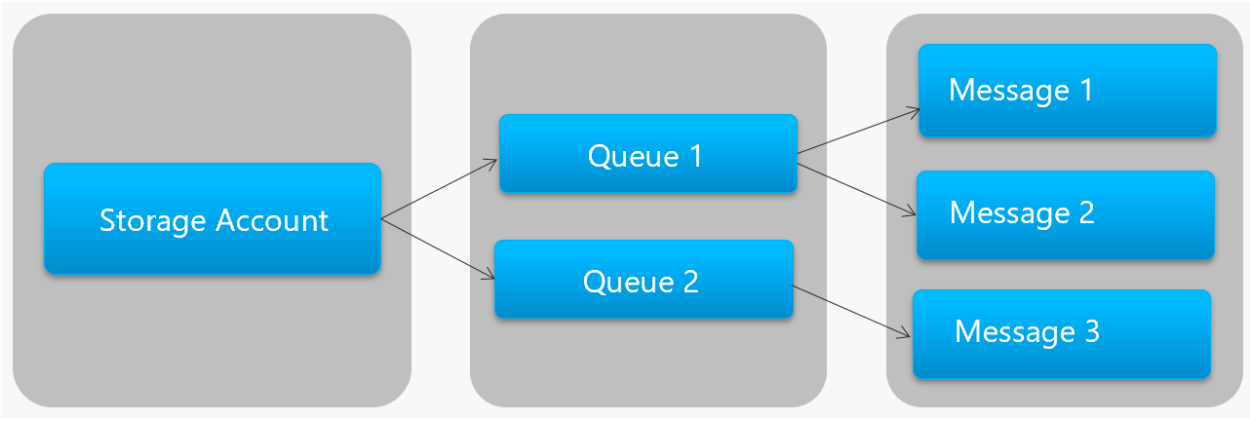


Figura 4-13. Jerarquía de colas de Azure Storage

En la imagen anterior, fíjese en cómo las colas de Storage almacenan los mensajes en la cuenta de Azure Storage subyacente.

Microsoft proporciona a los desarrolladores varias bibliotecas cliente y servidor para el procesamiento de colas de Storage. La mayoría de las plataformas principales son compatibles, incluidas .NET, JavaScript, Ruby, Python y Go. Los desarrolladores nunca deben comunicarse directamente con estas bibliotecas. Si lo hacen, el código del microservicio se acoplará estrechamente al servicio de colas de Azure Storage. Es mejor aislar los detalles de implementación de la API, incluir una capa de intermediación, o

una API intermedia, que exponga operaciones genéricas y que encapsule la biblioteca en concreto. Este acoplamiento flexible permite cambiar un servicio de puesta en cola por otro sin tener que realizar cambios en el código del servicio principal.

Las colas de Azure Storage son una opción económica para implementar la mensajería de comandos en las aplicaciones nativas de nube, especialmente si el tamaño de una cola va a superar los 80 GB o basta con un conjunto de características simple. Solo se paga por el almacenamiento de los mensajes; no hay cargos fijos por hora.

Azure Service Bus Queues

Si existen unos requisitos de mensajería más complejos, considere la posibilidad de usar colas de Azure Service Bus.

Azure Service Bus, que se cimenta en una infraestructura de mensajes muy sólida, admite un *modelo de mensajería asincrónica*. Los mensajes se almacenan de forma confiable en un agente (la cola) hasta que el consumidor los recibe. La cola garantiza una entrega de mensajes de tipo FIFO (el primero en entrar es el primero en salir), respetando el orden en el que los mensajes se agregaron a la cola.

El tamaño de un mensaje puede ser mucho mayor, hasta 256 KB. Los mensajes se conservan en la cola durante un período de tiempo ilimitado. Service Bus no solo admite llamadas basadas en HTTP, sino que también proporciona plena compatibilidad con el protocolo AMQP. AMQP es un estándar abierto entre proveedores que admite un protocolo binario y un mayor grado de confiabilidad.

Service Bus proporciona un amplio conjunto de características, incluida la compatibilidad con transacciones y una característica de detección de duplicados. La cola garantiza "una entrega como máximo" por mensaje. Descarta automáticamente un mensaje que ya se ha enviado. Si un productor tiene dudas, puede reenviar el mismo mensaje, y Service Bus solo procesará una copia. La detección de duplicados nos libera de tener que crear más infraestructura.

Dos características empresariales más son las particiones y las sesiones. Una cola de mensajes de Service Bus al uso se controla por medio de un único agente de mensajes y se almacena en un único almacén de mensajes. Sin embargo, con las particiones de Service Bus la cola de mensajes se reparte entre varios agentes de mensajes y varios almacenes de mensajes. El rendimiento general ya no está limitado por el rendimiento de un agente de mensajes o almacén de mensajería en concreto. Una interrupción temporal de un almacén de mensajería no hace que una cola particionada deje de estar disponible.

Las sesiones de Service Bus proporcionan una manera de agrupar mensajes relacionados. Imaginemos un escenario de flujo de trabajo en el que los mensajes se deben procesar juntos y la operación debe completarse al final. Para sacar partido de ellas, las sesiones deben habilitarse explícitamente en la cola, y cada mensaje relacionado debe contener el mismo identificador de sesión.

Pero esto hay algunas salvedades importantes: el tamaño de las colas de Service Bus está limitado a 80 GB, que es mucha menos capacidad que las colas de Storage. Además, las colas de Service Bus incurren en un coste base y un cargo por operación.

En la figura 4-14 se describe a grandes rasgos la arquitectura de una cola de Service Bus.

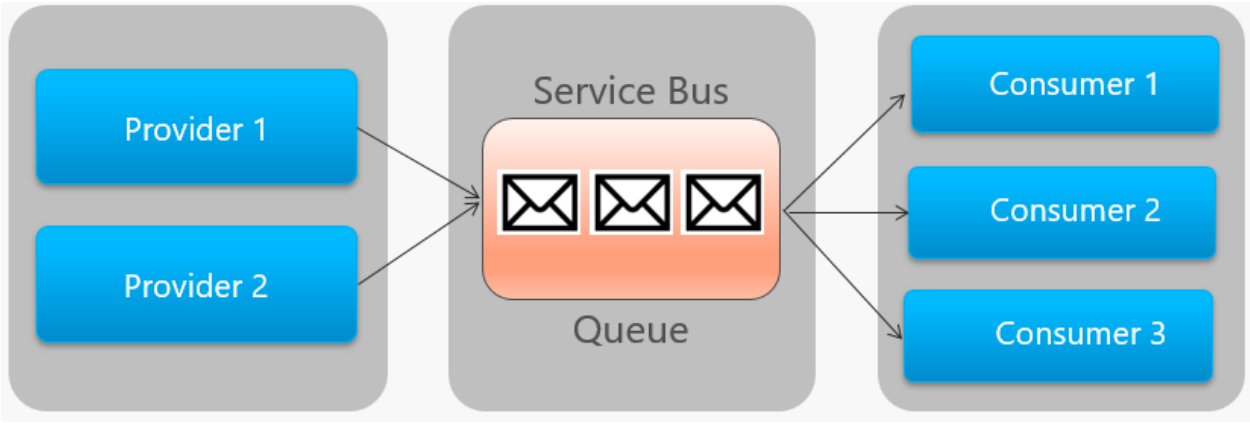


Figura 4-14. Cola de Service Bus

En la imagen anterior, observe la relación de punto a punto. Dos instancias del mismo proveedor ponen en cola mensajes en una misma cola de Service Bus. Solo una de las tres instancias de consumidor de la derecha consume cada mensaje. Analicemos ahora cómo implementar la mensajería cuando hay distintos consumidores interesados en el mismo mensaje.

Events

La cola de mensajes es una manera eficaz de implementar la comunicación, en la que un productor puede enviar de forma asincrónica un mensaje a un consumidor. Pero, ¿qué ocurre cuando *muchos consumidores diferentes* están interesados en el mismo mensaje? Una cola de mensajes dedicada para cada consumidor no escalaría bien y sería difícil de administrar.

Para abordar este escenario, pasaremos al tercer tipo de interacción de mensajes, el *evento*. Un microservicio anuncia que se ha producido una acción. Otros microservicios, si están interesados, reaccionan a la acción o al evento. Esto también se conoce como estilo arquitectónico controlado por eventos.

Los eventos consisten en un proceso de dos pasos. Ante un cambio de estado determinado, un microservicio publica un evento en un agente de mensajes, lo que hace que esté disponible para cualquier otro microservicio interesado. El microservicio interesado es notificado mediante la suscripción al evento en el agente de mensajes. Para implementar la comunicación basada en eventos se usa el patrón publicador/suscriptor.

En la figura 4-15 se muestra un microservicio de cesta de la compra que publica un evento y que tiene otros dos microservicios suscritos a él.

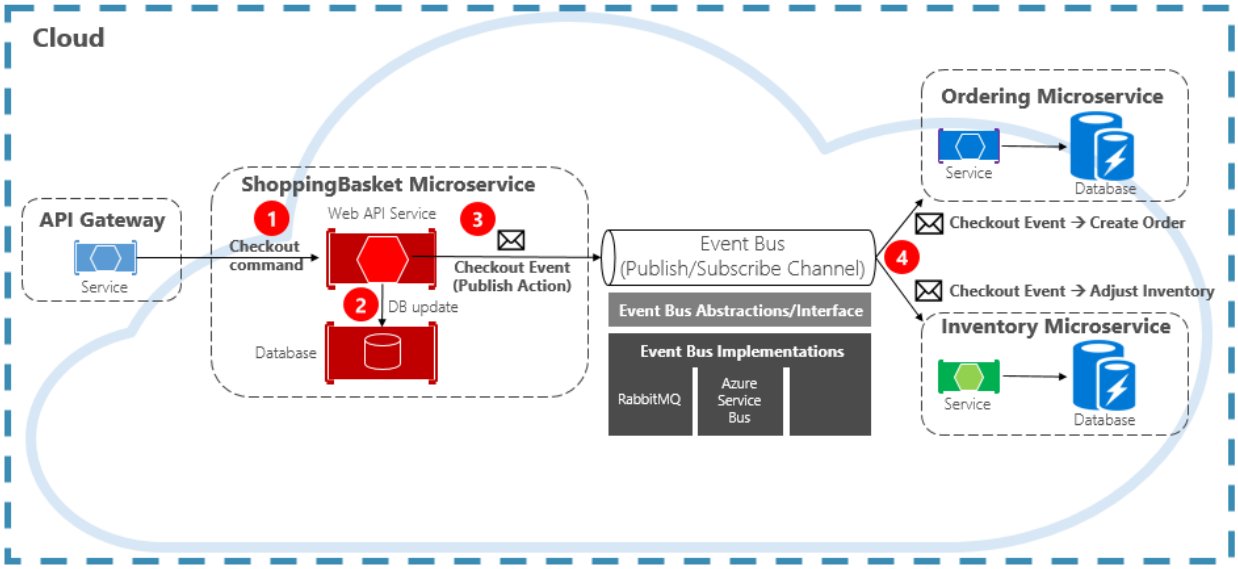


Figura 4-15. Mensajería controlada por eventos

Fíjese en el componente de *bus de eventos* que hay en el centro del canal de comunicación. Es una clase personalizada que encapsula el agente de mensajes y lo desacopla de la aplicación subyacente. Los microservicios de pedidos e inventario operan el evento de forma independiente sin tener conocimiento el uno del otro, ni tampoco del microservicio de cesta de la compra. Cuando el evento registrado se publica en el bus de eventos, actúan como corresponda.

Con los eventos, pasamos de la tecnología de puesta en cola a los *temas*. Un tema es similar a una cola, pero admite un patrón de mensajería de uno a varios. Un microservicio publica un mensaje. Varios microservicios de suscripción pueden decidir recibir ese mensaje y actuar en consecuencia. En la figura 4-16 se muestra una arquitectura de temas.

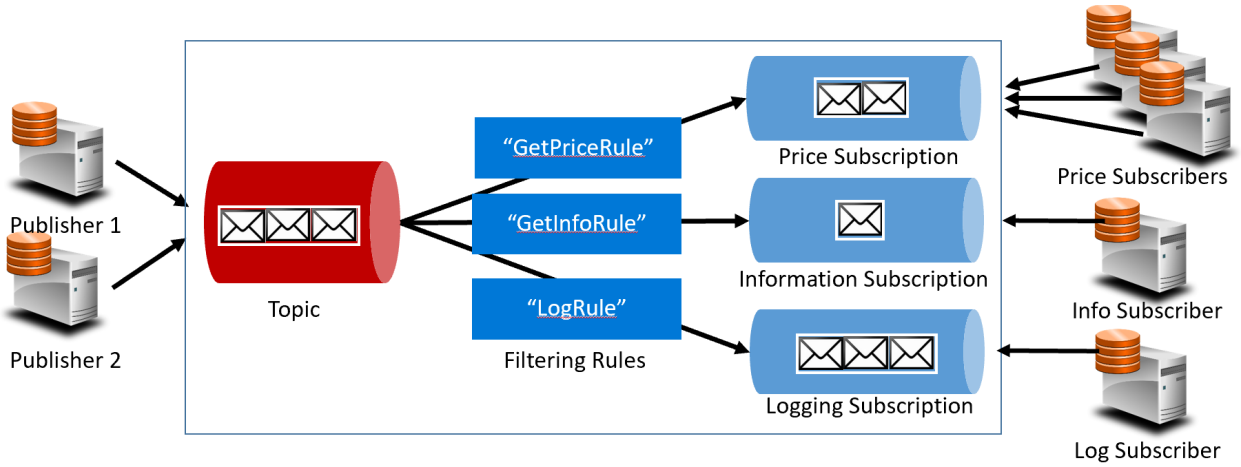


Figura 4-16. Arquitectura de temas

En la imagen anterior, los publicadores envían mensajes al tema. Al final, los suscriptores reciben mensajes de las suscripciones. En el medio, el tema reenvía mensajes a las suscripciones siguiendo un conjunto de reglas (los cuadros de color azul oscuro). Las reglas actúan como un filtro que reenvía mensajes específicos a una suscripción. Aquí, se enviaría un evento "GetPrice" a las suscripciones de precio y de registro, ya que la suscripción de registro ha elegido recibir todos los mensajes. También se enviaría un evento "GetInfo" a las suscripciones de información y de registro.

La nube de Azure admite dos servicios de temas diferentes: temas de Azure Service Bus y Azure Event Grid.

Azure Service Bus Topics

Los temas de Azure Service Bus se basan en el mismo modelo sólido de mensajes asincrónicos que las colas de Azure Service Bus. Un tema puede recibir mensajes de varios publicadores independientes y enviar mensajes a hasta 2000 suscriptores. Las suscripciones se pueden agregar o quitar dinámicamente en tiempo de ejecución, sin detener el sistema ni volver a crear el tema.

Muchas características avanzadas de las colas de Azure Service Bus también están disponibles en los temas, como la compatibilidad con la detección de duplicados y las transacciones. Los temas de Service Bus se controlan de forma predeterminada mediante un único agente de mensajes y se almacenan en un único almacén de mensajes. Sin embargo, las particiones de Service Bus escalan los temas y permiten repartirlos entre muchos agentes de mensajes y almacenes de mensajes.

La entrega de mensajes programada etiqueta un mensaje con una hora específica para procesarlo. El mensaje no aparecerá en el tema antes de esa hora. El aplazamiento de mensajes permite aplazar la recuperación de un mensaje en un momento posterior. Ambos se suelen usar en escenarios de procesamiento de flujos de trabajo en los que

las operaciones se procesan en un orden determinado. El procesamiento de los mensajes recibidos se puede posponer hasta que se haya completado el trabajo anterior.

Los temas de Service Bus son una tecnología sólida y de probada eficacia para habilitar la comunicación entre publicador y suscriptor en los sistemas nativos de nube.

Azure Event Grid

Aunque Azure Service Bus es un agente de mensajería cuya compatibilidad con un conjunto completo de características empresariales está más que probada, Azure Event Grid es el nuevo niño bonito.

A primera vista, Event Grid puede parecer un sistema más de mensajería basada en temas. Sin embargo, es diferente en muchas vertientes. Se centra en las cargas de trabajo controladas por eventos, lo que permite el procesamiento de eventos en tiempo real, una profunda integración con Azure y una infraestructura sin servidor de plataforma abierta. Está diseñado para las aplicaciones nativas de nube y sin servidor de hoy día.

Como un *backplane de eventos* centralizado, o canalización, Event Grid reacciona a los eventos dentro de los recursos de Azure y desde sus propios servicios.

Las notificaciones de eventos se publican en un tema de Event Grid, que, a su vez, enruta cada evento a una suscripción. Los suscriptores se asignan a suscripciones y consumen los eventos. Al igual que Service Bus, Event Grid admite un *modelo de suscriptor filtrado* según el cual cada suscripción establece las reglas relativas a los eventos que quiere recibir. El rendimiento de Event Grid es tremendamente rápido, con una garantía de 10 millones de eventos por segundo, lo que permite la entrega casi en tiempo real, mucho más de lo que Azure Service Bus es capaz de generar.

Algo genial de Event Grid es su profunda integración en el tejido de la infraestructura de Azure. Un recurso de Azure, como Cosmos DB, puede publicar eventos integrados directamente en otros recursos de Azure interesados, sin necesidad de código personalizado. Event Grid puede publicar eventos desde una suscripción de Azure, un grupo de recursos o un servicio, lo que confiere a los desarrolladores un control específico sobre el ciclo de vida de los recursos de nube. Sin embargo, Event Grid no se limita a Azure, sino que se trata una plataforma abierta que puede consumir eventos HTTP personalizados publicados desde aplicaciones o servicios de terceros, así como enrutar eventos a suscriptores externos.

Al publicar eventos nativos de recursos de Azure y suscribirse a ellos, no se requiere codificación alguna. Con una configuración sencilla, los eventos de un recurso de Azure

se pueden integrar en otro haciendo uso de la "fontanería" ya integrada de los temas y las suscripciones. En la figura 4-17 se muestra la anatomía de Event Grid.

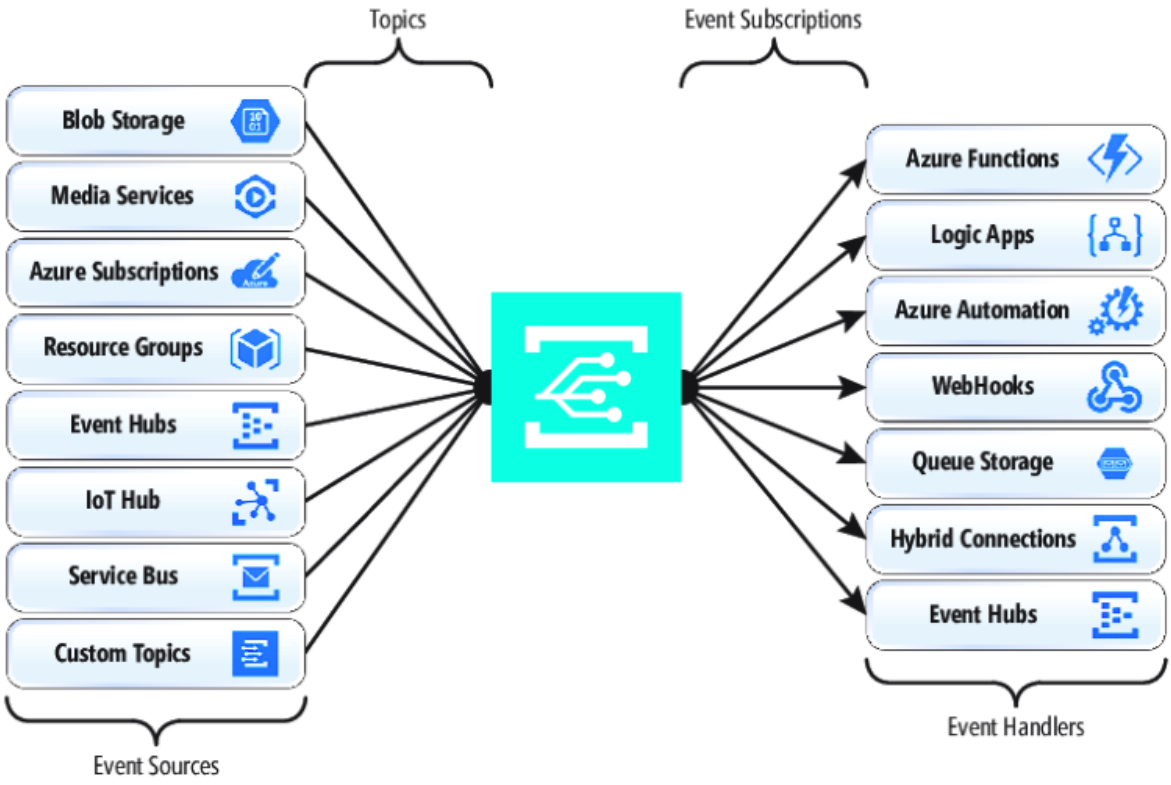


Figura 4-17. Anatomía de Event Grid

Una diferencia importante entre Event Grid y Service Bus es el *patrón de intercambio de mensajes* subyacente.

Service Bus implementa un *modelo de extracción* más antiguo según el cual el suscriptor descendente sondea activamente la suscripción a temas para ver si hay mensajes nuevos. El lado bueno de este enfoque es que el suscriptor tiene control total del ritmo al que procesa los mensajes. Controla cuándo y cuántos mensajes se van a procesar en un momento dado. Los mensajes no leídos permanecen en la suscripción hasta que se procesan. Una limitación significativa es la latencia entre el momento en que el evento se genera y la operación de sondeo que extrae ese mensaje al suscriptor para su procesamiento. Asimismo, la sobrecarga de este sondeo constante en pos del siguiente evento consume recursos y dinero.

Pero Event Grid es distinto. Implementa un *modelo de inserción* según el cual los eventos se envían a controladores de eventos en el momento en que se reciben, de modo que la entrega de eventos tiene lugar casi en tiempo real. También reduce el coste, ya que el servicio se desencadena solo cuando es necesario consumir un evento, no continuamente, como ocurre con el sondeo. Dicho esto, un controlador de eventos debe controlar la carga entrante y proporcionar mecanismos de limitación para no sentirse sobrepasado. Muchos servicios de Azure que consumen estos eventos, como

Azure Functions y Logic Apps, proporcionan funcionalidades de escalado automático para controlar el aumento de las cargas.

Event Grid es un servicio de nube y sin servidor totalmente administrado. Escala dinámicamente en función del tráfico, y los cargos derivados corresponden a un uso real, no a una capacidad adquirida previamente. Las primeras 100 000 operaciones al mes son gratis: operaciones como la entrada de eventos (notificaciones de eventos entrantes), los intentos de entrega de suscripciones, las llamadas de administración y el filtrado por asunto. Con una disponibilidad del 99,99 %, Event Grid garantiza la entrega de un evento en un plazo de 24 horas, con funcionalidad de reintento integrada en caso de que la entrega no se materialice. Los mensajes no entregados se pueden mover a una cola de "mensajes fallidos" para su resolución. A diferencia de Azure Service Bus, Event Grid está optimizado para alcanzar un rendimiento rápido, y no admite características como la mensajería ordenada, las transacciones ni las sesiones.

Transmisión de mensajes en la nube de Azure

Azure Service Bus y Event Grid proporcionan una gran compatibilidad con aplicaciones que exponen eventos únicos discretos, como cuando un documento nuevo se ha insertado en Cosmos DB. Pero, ¿qué ocurre si el sistema nativo de nube necesita procesar una *secuencia de eventos relacionados*? Las secuencias de eventos son más complicadas. Suelen ordenarse por tiempo, están interrelacionadas y deben procesarse como un todo.

Azure Event Hubs es una plataforma de transmisión de datos y un servicio de ingesta de eventos que recopila, transforma y almacena eventos. Está optimizado para capturar datos de transmisión, como las continuas notificaciones de eventos emitidas desde un contexto de telemetría. Este servicio es tremendamente escalable y es capaz de almacenar y procesar millones de eventos por segundo. Ilustrado en la figura 4-18, suele actuar como puerta de entrada de una canalización de eventos, lo que separa la secuencia de ingesta del consumo de eventos.

Figura 4-18. Centro de eventos de Azure

Event Hubs admite una latencia baja y la retención de tiempo configurable. A diferencia de las colas y los temas, Event Hubs conserva los datos de los eventos una vez leídos por un consumidor. Esta característica permite que otros servicios analíticos de datos, tanto internos como externos, reproduzcan los datos para su posterior análisis. Los eventos almacenados en un centro de eventos solo se eliminan una vez ha expirado el período de retención, cuyo valor predeterminado es un día, pero esto se puede configurar.

Event Hubs admite protocolos comunes de publicación de eventos, incluidos HTTPS y AMQP. También admite Kafka 1.0. Las aplicaciones de Kafka existentes pueden comunicarse con Event Hubs a través del protocolo de Kafka, lo que supone una alternativa a tener que administrar clústeres de Kafka de gran tamaño. Muchos sistemas nativos de nube y de código abierto usan Kafka.

Event Hubs implementa la transmisión de mensajes a través de un modelo de consumidor de particiones, en el que cada consumidor lee solo un subconjunto específico, o partición, de la secuencia de mensajes. Este patrón permite una gran escala horizontal para procesamiento de eventos y proporciona otras características centradas en secuencias que no están disponibles en colas y temas. Una partición es una secuencia ordenada de eventos que se mantiene en un centro de eventos. A medida que llegan eventos más recientes, se agregan al final de esta secuencia. En la figura 4-19 se ilustran las particiones de Event Hubs.

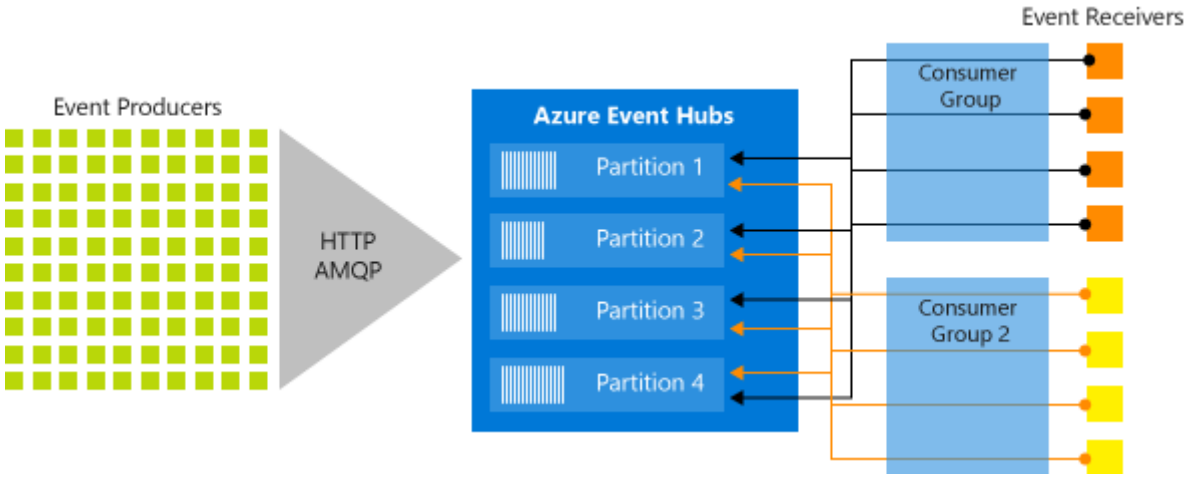


Figura 4-19. Particiones de Event Hubs

En lugar de leer todos el mismo recurso, cada grupo de consumidores lee un subconjunto o partición de la secuencia de mensajes.

Azure Event Hubs puede ser una solución sólida y asequible para las aplicaciones nativas de nube que deben transmitir un gran número de eventos.

Anterior Siguiente

gRPC

Artículo • 14/12/2023

Sugerencia

Este contenido es un extracto del libro electrónico “Architecting Cloud Native .NET Applications for Azure” (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Hasta ahora en este libro, nos hemos centrado en la comunicación basada en REST. Hemos visto que REST es un estilo de arquitectura flexible que define las operaciones basadas en CRUD en los recursos de entidad. Los clientes interactúan con recursos a través de HTTP con un modelo de comunicación de solicitud-respuesta. Aunque REST es una tecnología de comunicación más reciente ampliamente implementada, gRPC ha ganado un enorme impulso en toda la comunidad nativa de la nube.

¿Qué es gRPC?

gRPC es un marco moderno de alto rendimiento que desarrolla el protocolo de llamada a procedimiento remoto (RPC) antiguo. A nivel de aplicación, gRPC simplifica la mensajería entre los clientes y los servicios back-end. Procedente de Google, gRPC es código abierto y forma parte del ecosistema Cloud Native Computing Foundation (CNCF) de ofertas nativas de nube. CNCF considera a gRPC un proyecto de incubación. El término incubación significa que los usuarios finales usan la tecnología en las aplicaciones de producción y el proyecto tiene un número correcto de colaboradores.

Una aplicación cliente de gRPC típica expondrá una función local en proceso que implementa una operación empresarial. En segundo plano, esa función local invoca otra

función en una máquina remota. Lo que parece ser una llamada local básicamente se convierte en una llamada fuera de proceso transparente a un servicio remoto. La asociación RPC abstrae la comunicación de red de punto a punto, la serialización y la ejecución entre equipos.

En las aplicaciones nativas de la nube, los desarrolladores suelen trabajar entre diversos lenguajes de programación, marcos y tecnologías. Esta *interoperabilidad* complica los contratos de mensajes y la fontanería necesaria para la comunicación multiplataforma. gRPC proporciona una "capa horizontal uniforme" que abstrae estas preocupaciones. Los desarrolladores crean código en su plataforma nativa centrada en la funcionalidad empresarial, mientras que gRPC controla la fontanería de la comunicación.

gRPC ofrece compatibilidad completa entre las pilas de desarrollo más populares, como Java, JavaScript, C#, Go, Swift y NodeJS.

Ventajas de gRPC

gRPC usa HTTP/2 como protocolo de transporte. Aunque es compatible con HTTP 1.1, HTTP/2 presenta muchas funcionalidades avanzadas:

- Un protocolo de trama binaria para el transporte de datos, a diferencia de HTTP 1.1, que se basa en texto.
- Compatibilidad con multiplexación para enviar varias solicitudes paralelas a través de la misma conexión: HTTP 1.1 limita el procesamiento a un mensaje de solicitud-respuesta a la vez.
- Comunicación bidireccional de dúplex completo para enviar tanto solicitudes de cliente como respuestas de servidor simultáneamente
- Streaming integrado que permite solicitudes y respuestas a flujos asincrónicos de grandes conjuntos de datos.
- Compresión de encabezados que reduce el uso de la red.

gRPC es ligero y de alto rendimiento. Puede ser hasta 8 veces más rápido que la serialización JSON con mensajes entre un 60 y un 80 % más pequeños. En la jerga de Microsoft Windows Communication Foundation (WCF), el rendimiento de gRPC supera la velocidad y eficacia de los enlaces NetTCP muy optimizados. A diferencia de NetTCP, que favorece la pila de Microsoft, gRPC es multiplataforma.

Búferes de protocolo

gRPC adopta una tecnología de código abierto denominada Búferes de protocolo. Estos búferes proporcionan un formato de serialización muy eficiente e independiente

de la plataforma para serializar mensajes estructurados que los servicios se envían. Con un lenguaje de definición de interfaz (IDL) multiplataforma, los desarrolladores definen un contrato de servicio para cada microservicio. El contrato, implementado como un archivo `.proto` basado en texto, describe los métodos, las entradas y las salidas de cada servicio. El mismo archivo de contrato se puede usar para clientes y servicios gRPC creados en diferentes plataformas de desarrollo.

Con el archivo proto, el compilador Protobuf, `protoc`, genera código de cliente y servicio para la plataforma de destino. El código incluye los siguientes componentes:

- Objetos fuertemente tipados, compartidos por el cliente y el servicio, que representan las operaciones del servicio y los elementos de datos de un mensaje.
- Una clase base fuertemente tipada con la fontanería de red necesaria que el servicio gRPC remoto puede heredar y extender.
- Código auxiliar de cliente que contiene la fontanería necesaria para invocar el servicio gRPC remoto.

En tiempo de ejecución, cada mensaje se serializa como una representación de Protobuf estándar y se intercambia entre el cliente y el servicio remoto. A diferencia de JSON o XML, los mensajes de Protobuf se serializan como bytes binarios compilados.

Compatibilidad con gRPC en .NET

gRPC se integra en el SDK de .NET Core 3.0 y versiones posteriores. Las siguientes herramientas lo admiten:

- Visual Studio 2022 con la carga de trabajo ASP.NET y de desarrollo web instalada
- Visual Studio Code
- La CLI `dotnet`

El SDK incluye herramientas para el enrutamiento de puntos de conexión, IoC integrado y registro. El servidor web Kestrel de código abierto admite conexiones HTTP/2. En la figura 4-20 se muestra una plantilla de Visual Studio 2022 que aplica scaffolding a un proyecto esqueleto de un servicio gRPC. Tenga en cuenta que .NET es totalmente compatible con Windows, Linux y macOS.

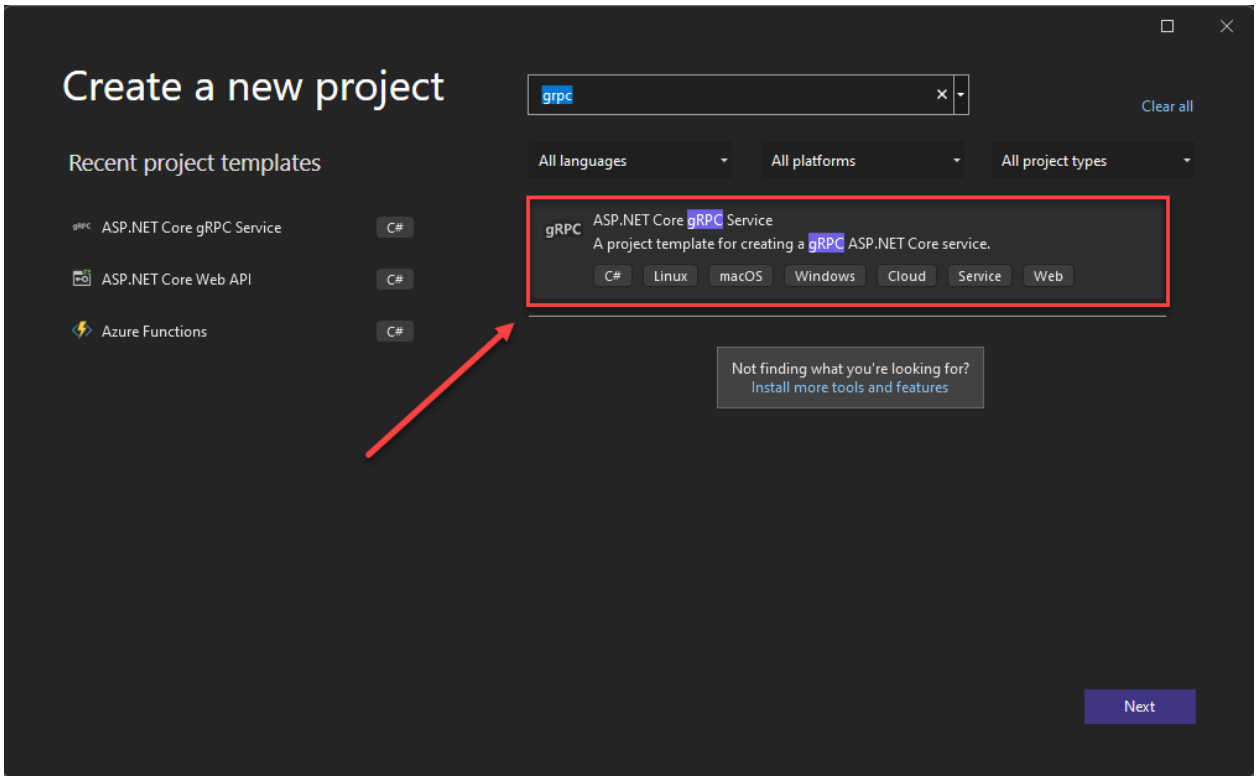


Figura 4-20. Compatibilidad con gRPC en Visual Studio 2022

En la figura 4-21 se muestra el servicio gRPC esqueleto generado a partir del scaffolding integrado incluido en Visual Studio 2022.

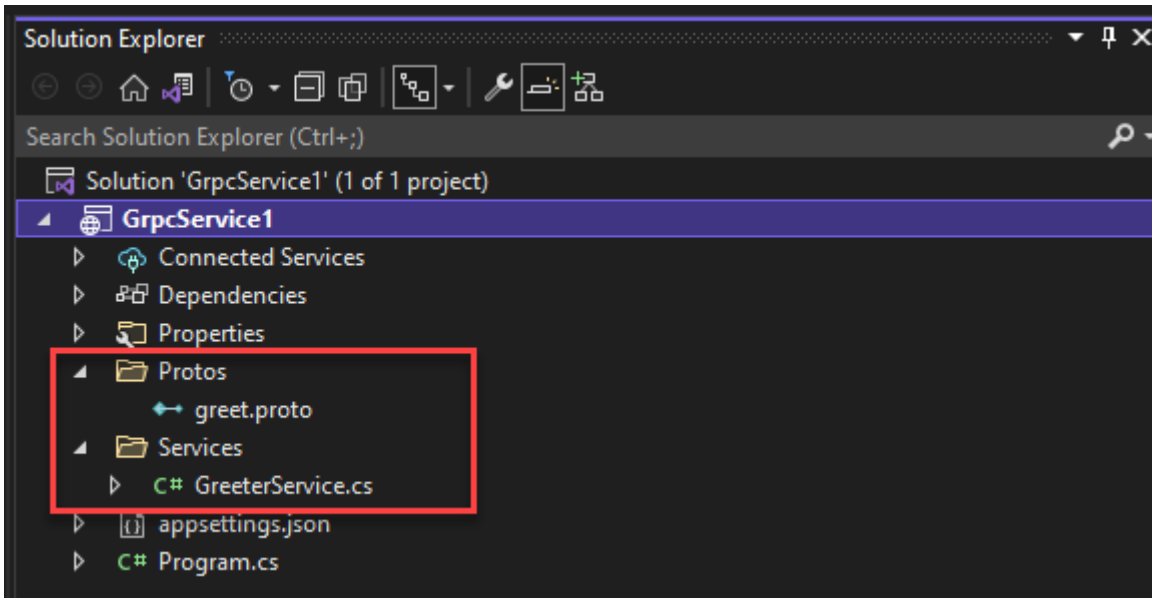


Figura 4-21. Proyecto gRPC en Visual Studio 2022

En la ilustración anterior, anote el archivo de descripción proto y el código de servicio. Como verá en breve, Visual Studio genera una configuración adicional tanto en la clase Startup como en el archivo de proyecto subyacente.

Uso de gRPC

Preferencia de gRPC en los siguientes escenarios:

- Comunicación sincrónica de microservicio a microservicio de back-end donde se requiere una respuesta inmediata para continuar el procesamiento.
- Entornos políglotas que necesitan admitir plataformas de programación mixtas.
- Baja latencia y comunicación de alto rendimiento donde el rendimiento es crítico.
- Comunicación en tiempo real de punto a punto: gRPC puede insertar mensajes en tiempo real sin sondeo y tiene una excelente compatibilidad con el streaming bidireccional.
- Entornos restringidos de red: los mensajes binarios de gRPC siempre son más pequeños que un mensaje JSON basado en texto equivalente.

En el momento de redactar este documento, gRPC se usa principalmente con servicios back-end. Los exploradores modernos no pueden proporcionar el nivel de control HTTP/2 necesario para admitir un cliente gRPC de front-end. Dicho esto, hay compatibilidad con gRPC-Web con .NET que permite la comunicación gRPC desde aplicaciones basadas en explorador creadas con JavaScript o tecnologías Blazor WebAssembly. gRPC-Web permite que una aplicación gRPC de ASP.NET Core admita características de gRPC en aplicaciones de explorador:

- Clientes fuertemente tipados y generados por código
- Mensajes de Protobuf compactos
- Streaming de servidor

Implementación de gRPC

La arquitectura de referencia de microservicios, eShop on Containers, de Microsoft, muestra cómo implementar servicios gRPC en aplicaciones .NET. En la figura 4-22 se presenta la arquitectura de back-end.

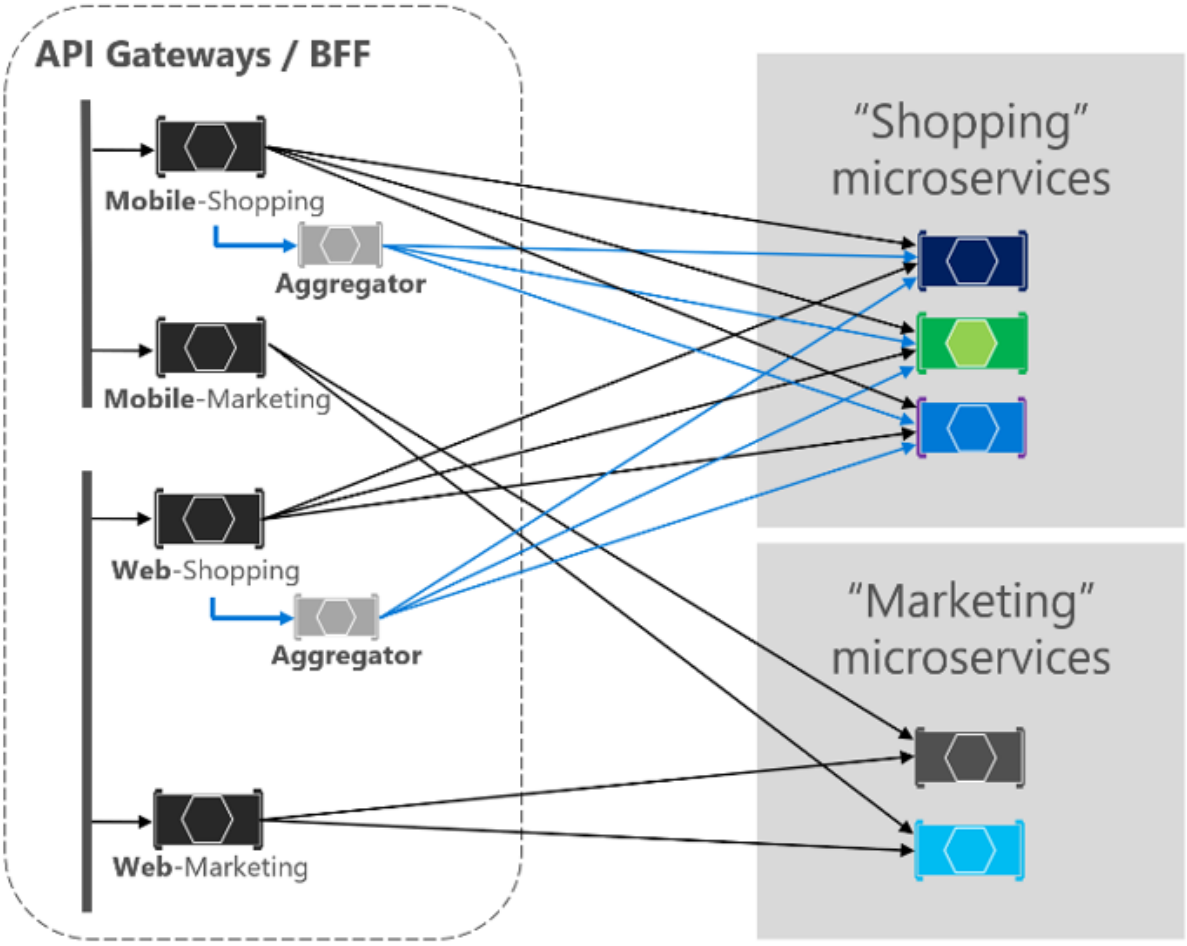


Figura 4-22. Arquitectura de back-end para eShop en contenedores

En la ilustración anterior, observe cómo eShop adopta el patrón Back-end para front-end (BFF) mediante la exposición de varias puertas de enlace de API. Anteriormente en este capítulo se ha analizado el patrón BFF. Preste mucha atención al microservicio Aggregator (en gris) que se encuentra entre la puerta de enlace de API de Web-Shopping y los microservicios Shopping de back-end. Aggregator recibe una única solicitud de un cliente, la envía a varios microservicios, agrega los resultados y los envía de vuelta al cliente solicitante. Normalmente, estas operaciones requieren una comunicación sincrónica para generar una respuesta inmediata. En eShop, las llamadas de back-end desde Aggregator se realizan mediante gRPC, como se muestra en la figura 4-23.

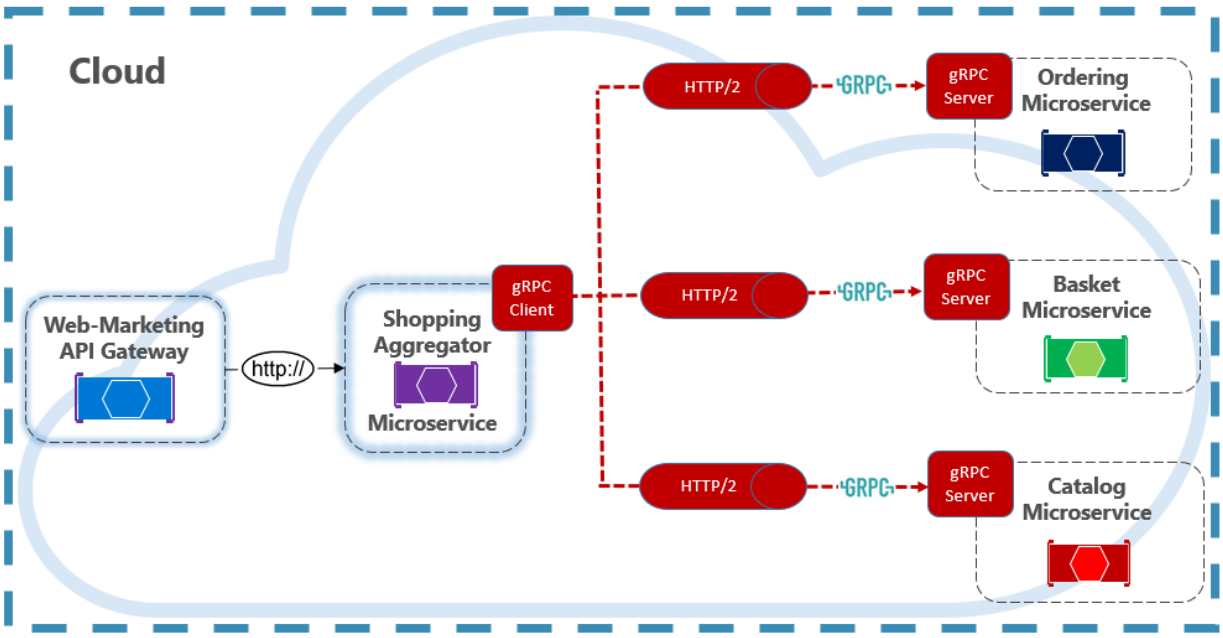


Figura 4-23. gRPC en eShop en contenedores

La comunicación gRPC requiere componentes cliente y servidor. En la ilustración anterior, observe cómo Shopping Aggregator implementa un cliente gRPC. El cliente realiza llamadas gRPC sincrónicas (en rojo) a microservicios de back-end, cada uno de los cuales implementa un servidor gRPC. Tanto el cliente como el servidor aprovechan las ventajas de la fontanería de gRPC integrada del SDK de .NET. Los *códigos auxiliares* del lado cliente proporcionan la fontanería para invocar llamadas gRPC remotas. Los componentes del lado servidor proporcionan la fontanería gRPC que las clases de servicio personalizadas pueden heredar y consumir.

Los microservicios que exponen una comunicación API RESTful y gRPC requieren varios puntos de conexión para administrar el tráfico. Abriría un punto de conexión que escucha el tráfico HTTP para las llamadas RESTful y otro para las llamadas gRPC. El punto de conexión gRPC debe configurarse para el protocolo HTTP/2 necesario para la comunicación gRPC.

Aunque nos esforzamos por desacoplar microservicios con patrones de comunicación asincrónicos, algunas operaciones requieren llamadas directas. gRPC debe ser la opción principal para la comunicación sincrónica directa entre microservicios. Su protocolo de comunicación de alto rendimiento, basado en HTTP/2 y búferes de protocolo, lo convierte en una opción perfecta.

Pensando en el futuro

De cara al futuro, gRPC seguirá ganando adeptos para los sistemas nativos de la nube. Las ventajas de rendimiento y la facilidad de desarrollo son convincentes. Sin embargo,

es probable que REST siga existiendo durante mucho tiempo. Destaca por las API expuestas públicamente y por motivos de compatibilidad con versiones anteriores.

Anterior Siguiente

Colaborar con nosotros en GitHub

El origen de este contenido se puede encontrar en GitHub, donde también puede crear y revisar problemas y solicitudes de incorporación de cambios. Para más información, consulte nuestra guía para colaboradores.

Comentarios de .NET

.NET es un proyecto de código abierto. Seleccione un vínculo para proporcionar comentarios:

Abrir incidencia con la documentación

Proporcionar comentarios sobre el producto

Infraestructura de comunicación de Service Mesh

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

A lo largo de este capítulo, hemos explorado los desafíos de la comunicación de microservicios. Hemos dicho que los equipos de desarrollo deben ser sensibles a cómo se comunican los servicios back-end entre sí. Cuanto menos sea la comunicación entre servicios, mejor. Sin embargo, la prevención no siempre es posible, ya que los servicios back-end a menudo se basan entre sí para completar las operaciones.

Hemos explorado diferentes enfoques para implementar la comunicación HTTP sincrónica y la mensajería asincrónica. En cada uno de los casos, el desarrollador se carga con la implementación de código de comunicación. El código de comunicación es complejo y requiere mucho tiempo. Las decisiones incorrectas pueden provocar problemas de rendimiento significativos.

Un enfoque más moderno para la comunicación de microservicios gira en torno a una tecnología nueva y en rápida evolución titulada *Service Mesh*. Una malla de servicio es una capa de infraestructura configurable con funcionalidades integradas para controlar la comunicación entre servicios, la resistencia y muchos problemas transversales. Mueve la responsabilidad de estos problemas fuera de los microservicios y a la capa de malla de servicio. La comunicación se abstrae de los microservicios.

Un componente clave de una malla de servicio es un proxy. En una aplicación nativa de la nube, una instancia de un proxy normalmente se coloca con cada microservicio. Mientras se ejecutan en procesos independientes, los dos están estrechamente vinculados y comparten el mismo ciclo de vida. Este patrón, conocido como patrón Sidecar, y se muestra en la figura 4-24.

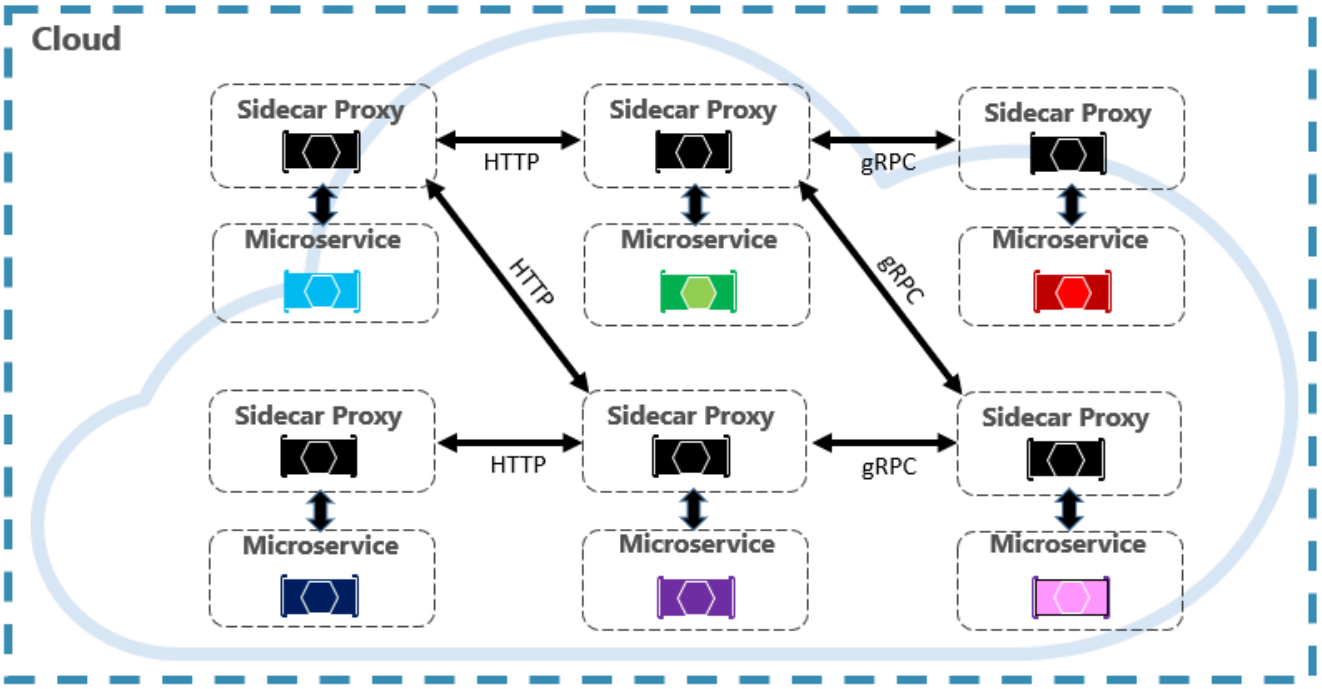


Figura 4-24. Malla de servicio con un coche lateral

Tenga en cuenta en la ilustración anterior cómo interceptan los mensajes un proxy que se ejecuta junto con cada microservicio. Cada proxy se puede configurar con reglas de tráfico específicas del microservicio. Comprende los mensajes y puede enrutarlos a través de sus servicios y del mundo exterior.

Junto con la administración de la comunicación entre servicios, el Service Mesh proporciona soporte para la detección de servicios y el equilibrio de carga.

Una vez configurada, una malla de servicio es muy funcional. La malla recupera un grupo correspondiente de instancias de un punto de conexión de detección de servicios. Envía una solicitud a una instancia de servicio específica, registrando la latencia y el tipo de respuesta del resultado. Elige la instancia más probable que devuelva una respuesta rápida en función de distintos factores, incluida la latencia observada para las solicitudes recientes.

Una malla de servicio administra los problemas de tráfico, comunicación y redes en el nivel de aplicación. Comprende los mensajes y las solicitudes. Normalmente, una malla de servicio se integra con un orquestador de contenedores. Kubernetes admite una arquitectura extensible en la que se puede agregar una malla de servicio.

En el capítulo 6, profundizamos en las tecnologías de Service Mesh, incluida una explicación sobre su arquitectura y las implementaciones de código abierto disponibles.

Resumen

En este capítulo, analizamos los patrones de comunicación nativos de la nube. Empezamos examinando cómo se comunican los clientes front-end con microservicios back-end. En el camino, hablamos sobre las plataformas de puerta de enlace de API y la comunicación en tiempo real. A continuación, examinamos cómo se comunican los microservicios con otros servicios back-end. Hemos examinado tanto la comunicación HTTP sincrónica como la mensajería asincrónica entre servicios. Tratamos gRPC, una tecnología emergente en el mundo nativo en la nube. Por último, presentamos una tecnología nueva y en rápida evolución titulada Service Mesh que puede simplificar la comunicación de microservicios.

Se hizo especial hincapié en los servicios administrados de Azure que pueden ayudar a implementar la comunicación en sistemas nativos de la nube:

- Azure Application Gateway
- Azure API Management
- Servicio Azure SignalR
- Colas de Azure Storage
- Azure Service Bus
- Azure Event Grid
- Centro de eventos de Azure

A continuación, pasaremos a datos distribuidos en sistemas nativos de nube y las ventajas y desafíos que presenta.

Referencias

- Microservicios de .NET: arquitectura para aplicaciones .NET en contenedor
- Diseño de la comunicación entre servicios para microservicios
- Azure SignalR Service, un servicio totalmente administrado para agregar funcionalidad en tiempo real
- Controlador de entrada de la puerta de enlace de API de Azure
- Documentación de gRPC
- Comparación de gRPC Services con API HTTP

- Compilación de gRPC Services con vídeo de .NET

Siguiente anterior

Patrones de datos nativos de la nube

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Como hemos visto en este libro, un enfoque nativo de la nube cambia la forma de diseñar, implementar y administrar aplicaciones. También cambia la forma de administrar y almacenar datos.

La figura 5-1 contrasta las diferencias.

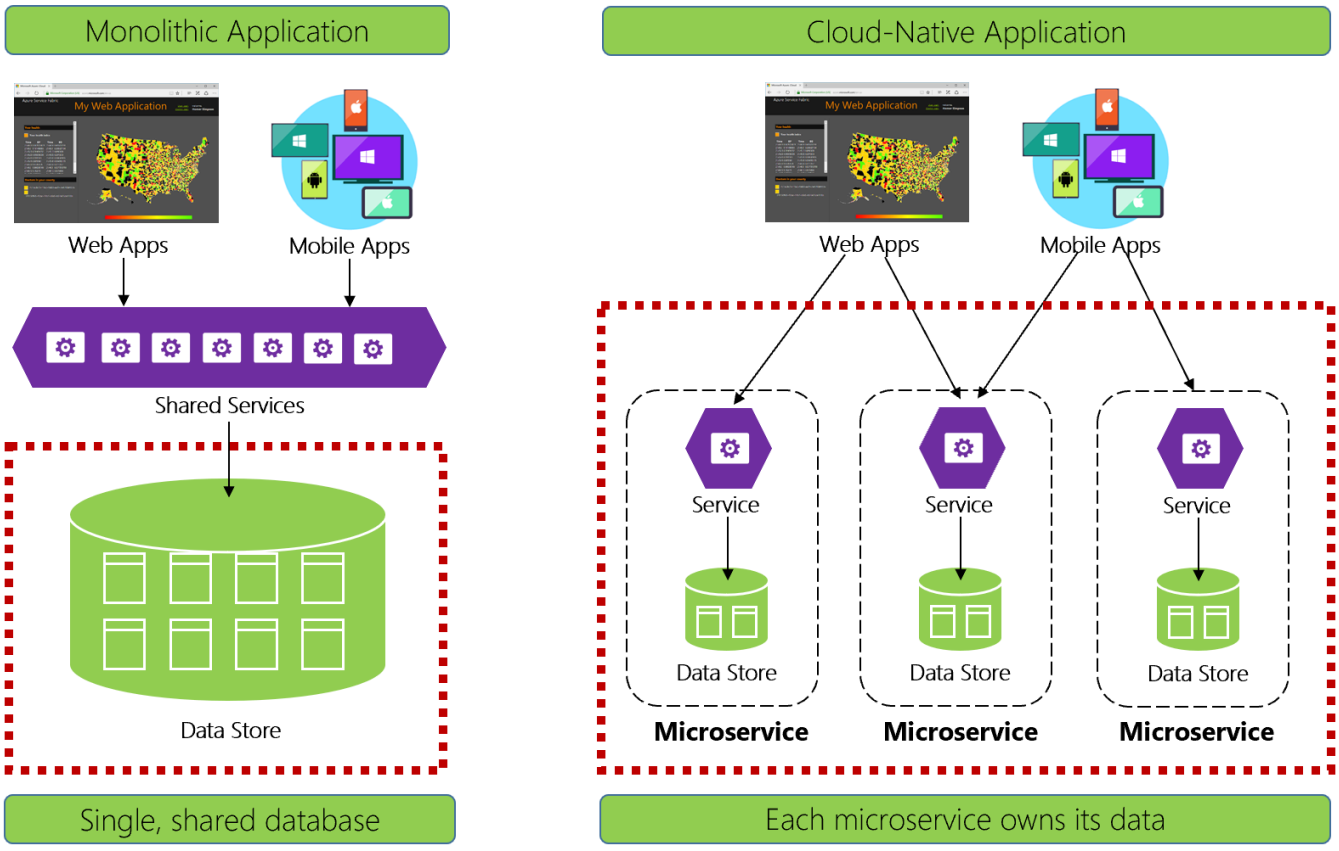


Figura 5-1. Administración de datos en aplicaciones nativas de nube

Los desarrolladores experimentados reconocerán fácilmente la arquitectura en el lado izquierdo de la figura 5-1. En esta *aplicación monolítica*, los componentes de servicio empresarial se agrupan en un nivel de servicios compartidos y comparten datos de una base de datos relacional única.

De muchas maneras, una base de datos única mantiene la administración de datos sencilla. La consulta de datos entre varias tablas es sencilla. Los cambios en los datos se actualizan juntos o se revierten todos. Las transacciones ACID garantizan una coherencia fuerte e inmediata.

Al diseñar para la nube nativa adoptamos un enfoque diferente. En el lado derecho de la figura 5-1, observe cómo la funcionalidad empresarial se separa en microservicios pequeños e independientes. Cada microservicio encapsula una funcionalidad empresarial específica y sus propios datos. La base de datos monolítica se descompone en un modelo de datos distribuido con muchas bases de datos más pequeñas, cada una de las cuales se alinea con un microservicio. Cuando se borra el humo, emergemos con un diseño que expone una *base de datos por microservicio*.

Base de datos por microservicio, ¿por qué?

Esta base de datos por microservicio proporciona muchas ventajas, especialmente para los sistemas que deben evolucionar rápidamente y admitir una escala masiva. Con este modelo...

- Los datos de dominio se encapsulan dentro del servicio
- El esquema de datos puede evolucionar sin afectar directamente a otros servicios
- Cada almacén de datos puede escalar de forma independiente
- Un error de almacén de datos en un servicio no afectará directamente a otros servicios

La separación de datos también permite a cada microservicio implementar el tipo de almacén de datos que está mejor optimizado para sus patrones de carga de trabajo, almacenamiento y lectura y escritura. Las opciones incluyen almacenes de datos relacionales, documentos, clave-valor e incluso basados en grafos.

La figura 5-2 presenta el principio de persistencia políglota en un sistema nativo en la nube.

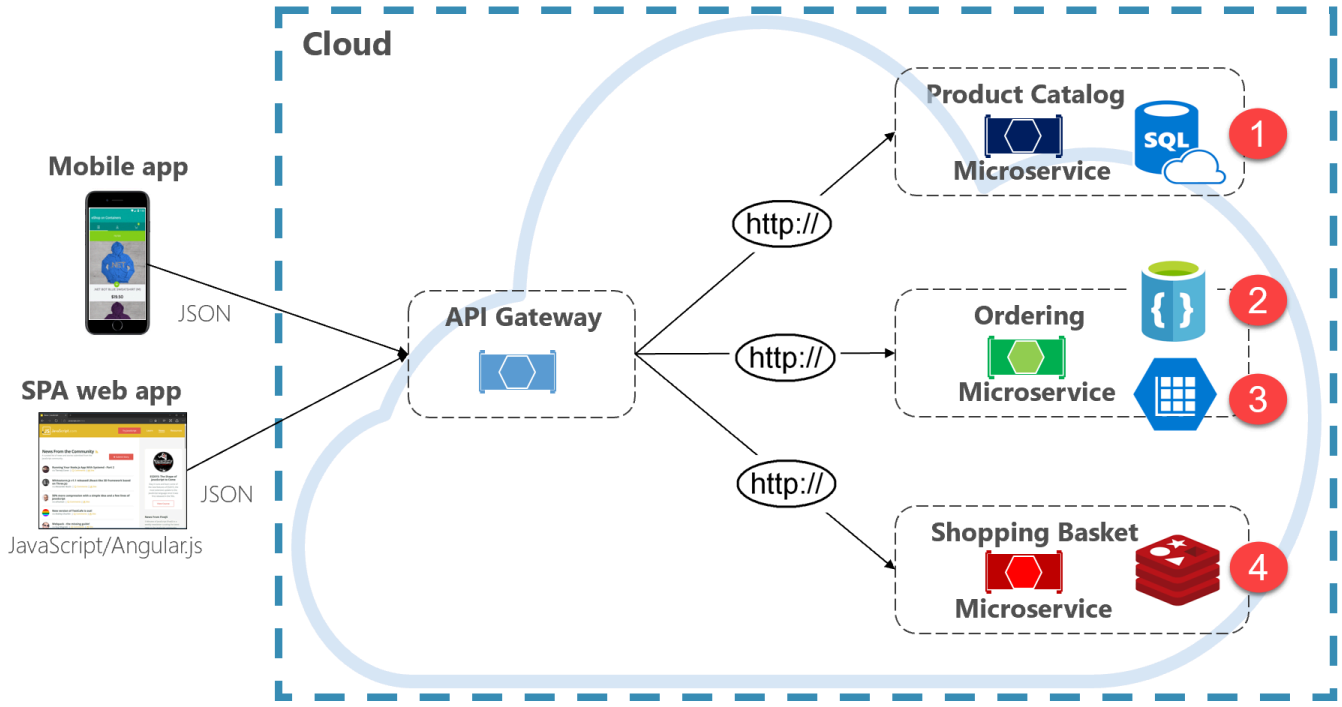


Figura 5-2. Persistencia de datos políglota

Tenga en cuenta en la ilustración anterior cómo cada microservicio admite un tipo diferente de almacén de datos.

- El microservicio del catálogo de productos consume una base de datos relacional para dar cabida a la estructura relacional enriquecida de sus datos subyacentes.
- El microservicio del carro de la compra consume una memoria caché distribuida que admite su sencillo almacén de datos de clave-valor.
- El microservicio de ordenación consume una base de datos de documentos NoSql para operaciones de escritura junto con un almacén de clave/valor altamente desnormalizado para dar cabida a grandes volúmenes de operaciones de lectura.

Aunque las bases de datos relacionales siguen siendo relevantes para los microservicios con datos complejos, las bases de datos NoSQL han ganado una popularidad considerable. Proporcionan gran escala y alta disponibilidad. Su naturaleza sin esquema permite a los

desarrolladores alejarse de una arquitectura de clases de datos tipadas y ORM que hacen que el cambio sea costoso y lento. Tratamos las bases de datos NoSQL más adelante en este capítulo.

Aunque la encapsulación de datos en microservicios independientes puede aumentar la agilidad, el rendimiento y la escalabilidad, también presenta muchos desafíos. En la sección siguiente, se describen estos desafíos junto con patrones y prácticas para ayudar a superarlos.

Consultas entre servicios

Aunque los microservicios son independientes y se centran en funcionalidades funcionales específicas, como el inventario, el envío o el pedido, con frecuencia requieren integración con otros microservicios. A menudo, la integración implica que un microservicio *consulte* a otro para los datos. En la figura 5-3 se muestra el escenario.

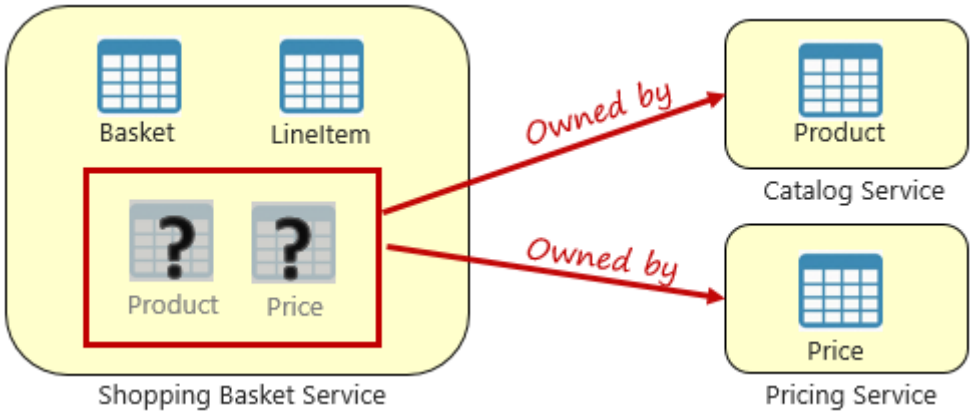


Figura 5-3. Consulta entre microservicios

En la ilustración anterior, vemos un microservicio de cesta de la compra que agrega un artículo a la cesta de la compra de un usuario. Aunque el almacén de datos de este microservicio contiene datos de artículos de cesta y línea, no mantiene los datos de productos ni precios. En su lugar, esos elementos de datos son propiedad del catálogo y los microservicios de precios. Este aspecto presenta un problema. ¿Cómo puede el microservicio de cesta de la compra agregar un producto a la cesta de la compra del usuario cuando no tiene datos de productos ni precios en su base de datos?

Una opción que se describe en el capítulo 4 es una llamada HTTP directa desde la cesta de la compra al catálogo y a los microservicios de precios. Sin embargo, en el capítulo 4, dijimos que las llamadas HTTP sincrónicas vinculan microservicios juntos, reduciendo su autonomía y disminuyendo sus ventajas arquitectónicas.

También podríamos implementar un patrón de solicitud-respuesta con colas entrantes y salientes independientes para cada servicio. Sin embargo, este patrón es complicado y requiere fontanería para correlacionar mensajes de solicitud y respuesta. Aunque desacopla las llamadas

a los microservicios del backend, el servicio que realiza la llamada todavía debe esperar sincrónicamente a que se complete la llamada. Congestión de red, errores transitorios o un microservicio sobrecargado pueden dar lugar a operaciones de larga duración e incluso con errores.

En su lugar, un patrón ampliamente aceptado para quitar dependencias entre servicios es el patrón de vista materializado, que se muestra en la figura 5-4.

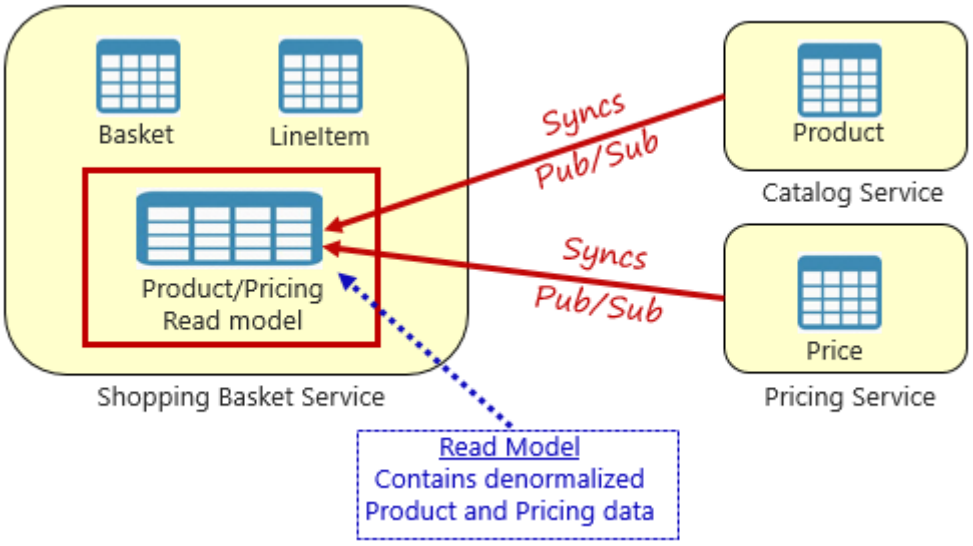


Figura 5-4. Patrón de vista materializado

Con este patrón, se coloca una tabla de datos local (conocida como *modelo de lectura*) en el servicio de carrito de compras. Esta tabla contiene una copia desnormalizada de los datos necesarios de los microservicios de producto y precios. Copiar los datos directamente en el microservicio de cesta de la compra elimina la necesidad de llamadas de servicio cruzado costosas. Con los datos locales para el servicio, se mejora el tiempo de respuesta y la confiabilidad del servicio. Además, tener su propia copia de los datos hace que el servicio de cesta de la compra sea más resistente. Si el servicio de catálogo no debería estar disponible, no afectaría directamente al servicio de cesta de la compra. La cesta de la compra puede seguir funcionando con los datos de su propia tienda.

El inconveniente de este enfoque es que ahora tiene datos duplicados en su sistema. Sin embargo, la duplicación *estratégica* de datos en sistemas nativos en la nube es una práctica establecida y no se considera un antipatrón o una mala práctica. Tenga en cuenta que *uno y solo un servicio* pueden poseer un conjunto de datos y tener autoridad sobre él. Deberá sincronizar los modelos de lectura cuando se actualice el sistema de registro. Normalmente, la sincronización se implementa mediante mensajería asincrónica con un patrón de publicación o suscripción, como se muestra en la figura 5.4.

Transacciones distribuidas

Aunque la consulta de datos entre microservicios es difícil, la implementación de una transacción en varios microservicios es aún más compleja. El desafío inherente de mantener la coherencia de los datos entre fuentes de datos independientes en distintos microservicios no puede ser subestimado. La falta de transacciones distribuidas en aplicaciones nativas de nube significa que debe administrar transacciones distribuidas mediante programación. Se mueve desde un mundo de *coherencia inmediata* a uno de *coherencia eventual*.

En la figura 5-5 se muestra el problema.

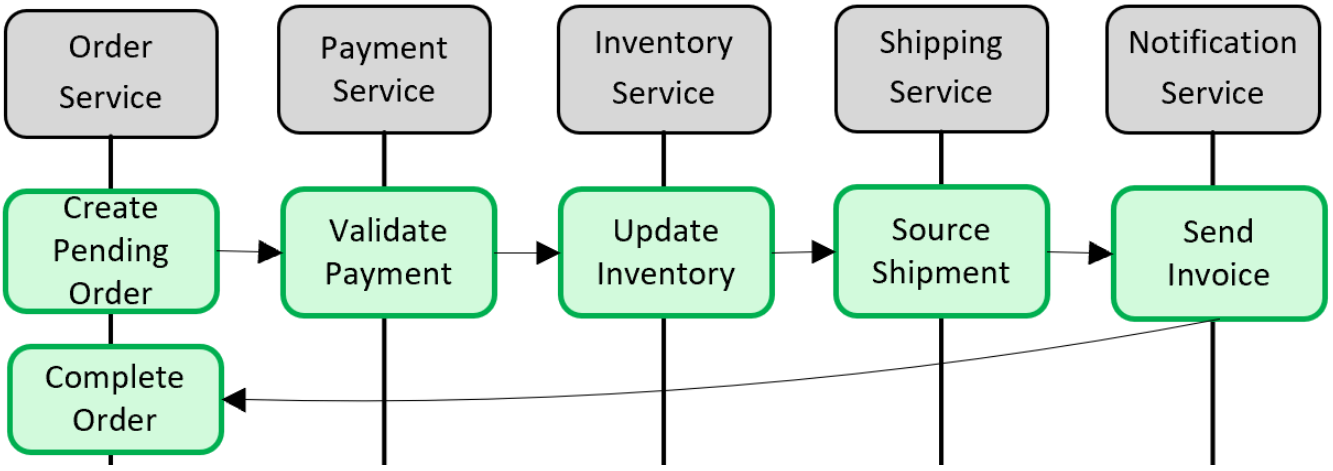


Figura 5-5. Implementación de una transacción entre microservicios

En la ilustración anterior, cinco microservicios independientes participan en una transacción distribuida que crea un pedido. Cada microservicio mantiene su propio almacén de datos e implementa una transacción local para su almacén. Para crear el pedido, la transacción local para *cada* microservicio individual debe tener éxito, de lo contrario *todas* deben anularse y revertir la operación. Aunque la compatibilidad transaccional integrada está disponible dentro de cada uno de los microservicios, no hay compatibilidad con una transacción distribuida que abarque los cinco servicios para mantener los datos coherentes.

En su lugar, debe construir esta transacción distribuida *mediante programación*.

Un patrón popular para agregar compatibilidad transaccional distribuida es el patrón Saga. Se implementa mediante la agrupación de transacciones locales mediante programación y la invocación secuencial de cada una. Si se produce un error en alguna de las transacciones locales, la Saga anula la operación e invoca un conjunto de transacciones de compensación. Las transacciones de compensación deshace los cambios realizados por las transacciones locales anteriores y restauran la coherencia de los datos. En la figura 5-6 se muestra una transacción fallida con el patrón Saga.

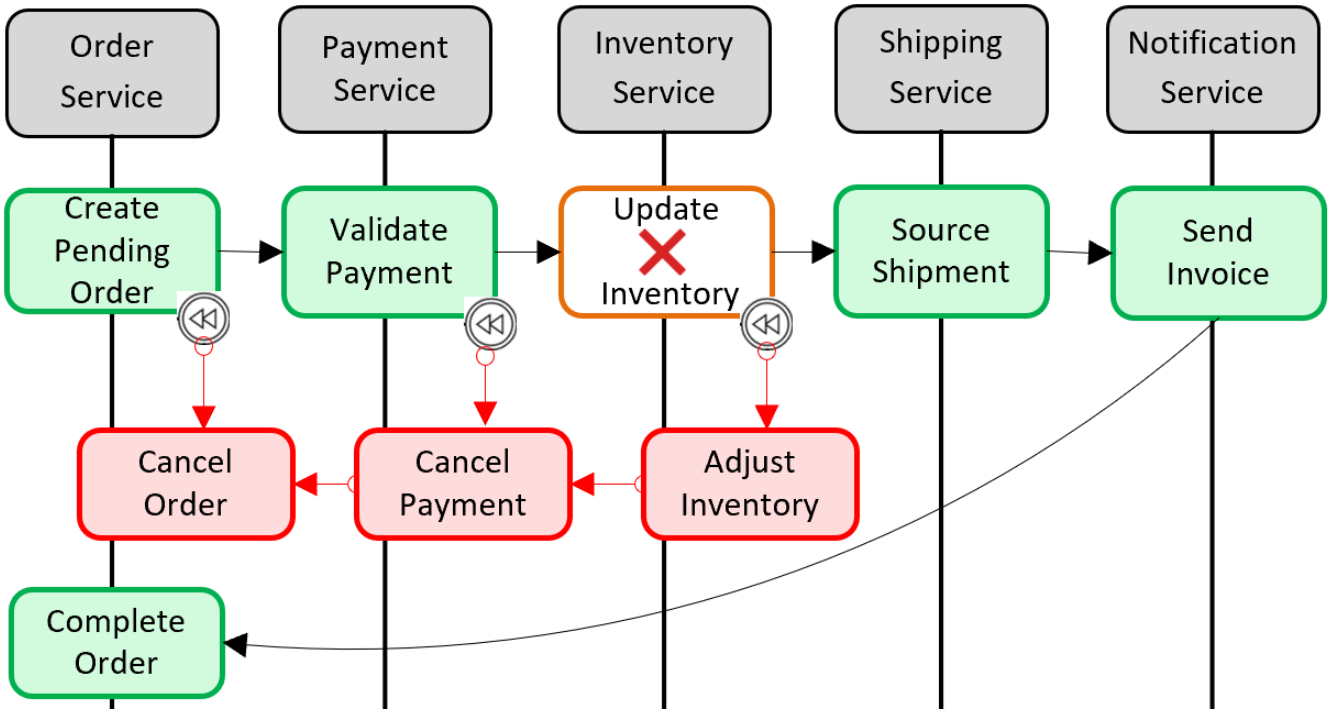


Figura 5-6. Revertir una transacción

En la ilustración anterior, se produjo un error en la operación *Actualizar inventario* en el microservicio Inventario. Saga invoca un conjunto de transacciones de compensación (en rojo) para ajustar los recuentos de inventario, cancelar el pago y el pedido, y devolver los datos de cada microservicio a un estado coherente.

Los patrones de Saga se suelen coreografiar como una serie de eventos relacionados o se orquestan como un conjunto de comandos relacionados. En el capítulo 4 analizamos el patrón del agregador de servicios que sería la base para una implementación orquestada de Saga. También discutimos la gestión de eventos junto con los temas de Azure Service Bus y Azure Event Grid, que serían la base para una implementación de saga coreografiada.

Datos de gran volumen

Las aplicaciones nativas de nube de gran tamaño suelen admitir requisitos de datos de gran volumen. En estos escenarios, las técnicas tradicionales de almacenamiento de datos pueden provocar cuellos de botella. En el caso de sistemas complejos que se implementan a gran escala, la segregación de responsabilidades de comandos y consultas (CQRS) y Event Sourcing pueden mejorar el rendimiento de la aplicación.

CQRS

CQRS, es un patrón arquitectónico que puede ayudar a maximizar el rendimiento, la escalabilidad y la seguridad. El patrón separa las operaciones que leen datos de esas operaciones que escriben datos.

En escenarios normales, se usan el mismo modelo de entidad y el objeto de repositorio de datos para las operaciones *de* lectura y escritura.

Sin embargo, un escenario de datos de gran volumen puede beneficiarse de modelos independientes y tablas de datos para lecturas y escrituras. Para mejorar el rendimiento, la operación de lectura podría consultar una representación altamente desnormalizada de los datos para evitar combinaciones de tablas repetitivas costosas y bloqueos de tabla. La operación de *escritura* , conocida como comando, se actualizaría con una representación totalmente normalizada de los datos que garantizarían la coherencia. A continuación, debe implementar un mecanismo para mantener sincronizadas ambas representaciones. Normalmente, cada vez que se modifica la tabla de escritura, publica un evento que replica la modificación en la tabla de lectura.

En la figura 5-7 se muestra una implementación del patrón CQRS.

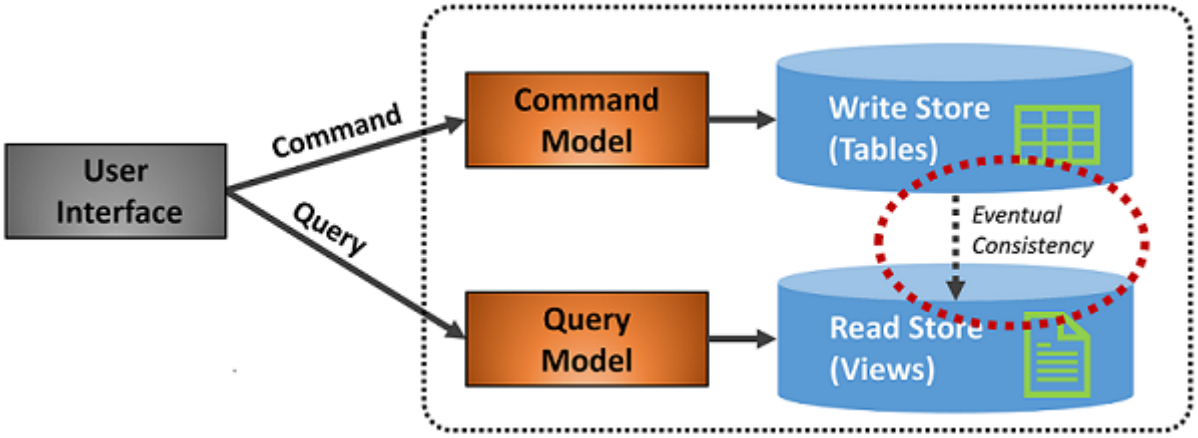


Figura 5-7. Implementación de CQRS

En la ilustración anterior, se implementan modelos de consulta y comandos independientes. Cada operación de escritura de datos se guarda en el almacén de escritura y, a continuación, se propaga al almacén de lectura. Preste atención a cómo funciona el proceso de propagación de datos en el principio de consistencia eventual . El modelo de lectura se sincroniza finalmente con el modelo de escritura, pero puede haber algún retraso en el proceso. En la sección siguiente se describe la coherencia final.

Esta separación permite que las lecturas y escrituras escalen independientemente. Las operaciones de lectura usan un esquema optimizado para consultas, mientras que las escrituras usan un esquema optimizado para actualizaciones. Las consultas de lectura se aplican a datos desnormalizados, mientras que la lógica empresarial compleja se puede aplicar al modelo de escritura. Además, podría imponer una seguridad más estricta en las operaciones de escritura que las que exponen las lecturas.

La implementación de CQRS puede mejorar el rendimiento de las aplicaciones para los servicios nativos de la nube. Sin embargo, resulta en un diseño más complejo. Aplique este

principio cuidadosa y estratégicamente a esas secciones de la aplicación nativa de la nube que se beneficiarán de ella. Para obtener más información sobre CQRS, consulte el libro de Microsoft Microservicios de .NET: Arquitectura para aplicaciones .NET en contenedor.

Aprovisionamiento de eventos

Otro enfoque para optimizar escenarios de datos de gran volumen implica Event Sourcing.

Normalmente, un sistema almacena el estado actual de una entidad de datos. Si un usuario cambia su número de teléfono, por ejemplo, el registro del cliente se actualiza con el nuevo número. Siempre sabemos el estado actual de una entidad de datos, pero cada actualización sobrescribe el estado anterior.

En la mayoría de los casos, este modelo funciona bien. Sin embargo, en sistemas de gran volumen, la sobrecarga del bloqueo transaccional y las operaciones de actualización frecuentes pueden afectar al rendimiento, la capacidad de respuesta y la escalabilidad del límite de la base de datos.

Event Sourcing adopta un enfoque diferente para capturar datos. Cada operación que afecta a los datos se conserva en un almacén de eventos. En lugar de actualizar el estado de un registro de datos, anexamos cada cambio a una lista secuencial de eventos anteriores, similar al libro de contabilidad de un contador. El Almacén de eventos se convierte en el sistema de registro de los datos. Se usa para propagar varias vistas materializadas dentro del contexto limitado de un microservicio. En la figura 5.8 se muestra el patrón.

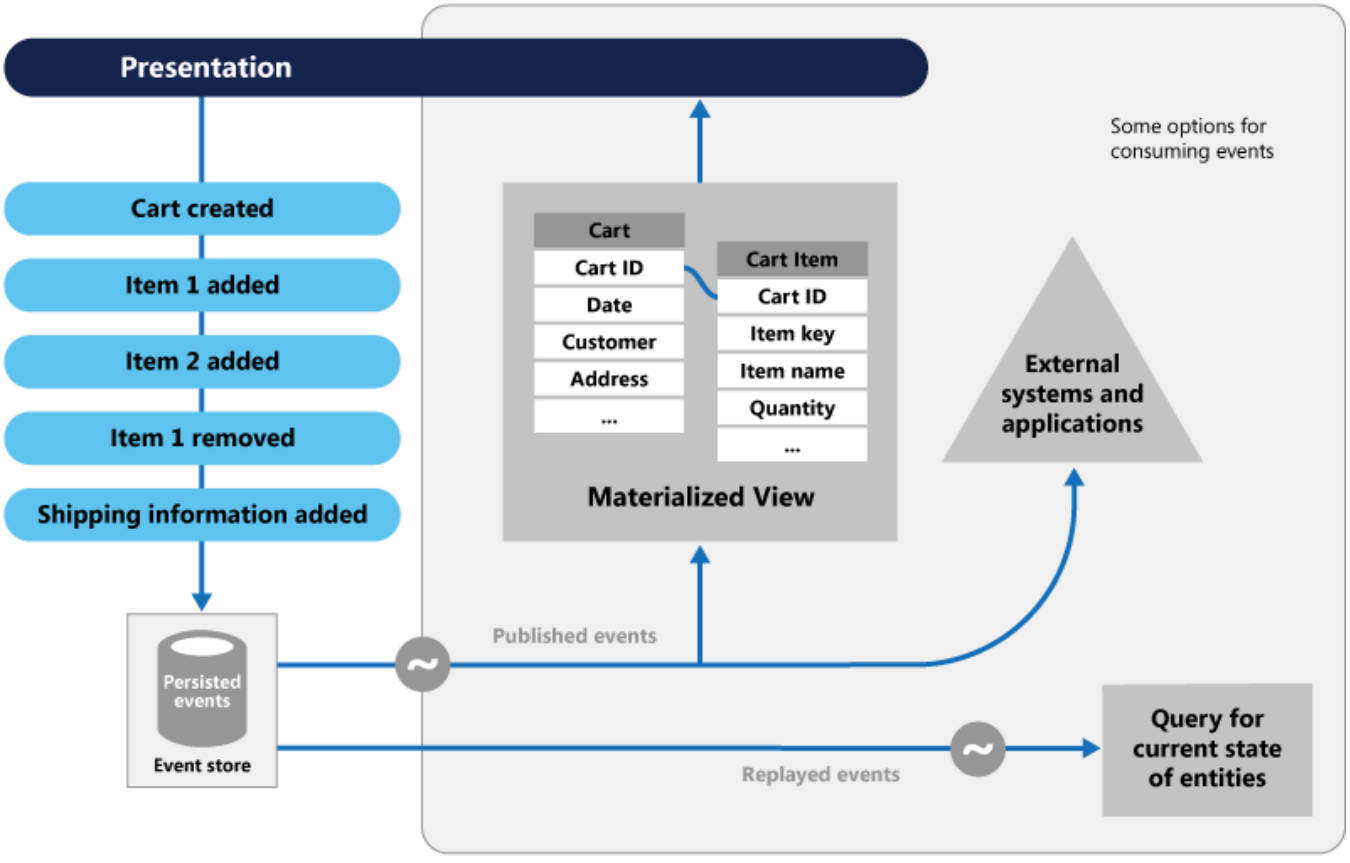


Figura 5-8. Aprovisionamiento de eventos

En la ilustración anterior, fíjate en cómo se añade cada entrada (en azul) para la cesta de compras de un usuario a un almacén de eventos subyacente. En la vista materializada adyacente, el sistema proyecta el estado actual reproduciendo todos los eventos asociados a cada carro de la compra. Esta vista, o modelo de lectura, se expone de nuevo a la interfaz de usuario. Los eventos también se pueden integrar con aplicaciones y sistemas externos o consultar para determinar el estado actual de una entidad. Con este enfoque, se mantiene el historial. Sabe no solo el estado actual de una entidad, sino también cómo alcanzó este estado.

En términos mecánicos, el aprovisionamiento de eventos simplifica el modelo de escritura. No hay actualizaciones ni eliminaciones. Anexar cada entrada de datos como un evento inmutable minimiza los conflictos de contención, bloqueo y simultaneidad asociados a bases de datos relacionales. La creación de modelos de lectura con el patrón de vista materializado permite desacoplar la vista del modelo de escritura y elegir el mejor almacén de datos para optimizar las necesidades de la interfaz de usuario de la aplicación.

Para este patrón, tome en consideración un almacén de datos que admita directamente el aprovisionamiento de eventos. Azure Cosmos DB, MongoDB, Cassandra, CouchDB y RavenDB son buenos candidatos.

Al igual que con todos los patrones y tecnologías, implemente estratégicamente y cuando sea necesario. Aunque el aprovisionamiento de eventos puede proporcionar un mayor rendimiento y escalabilidad, se produce a costa de la complejidad y una curva de aprendizaje.

Siguiente anterior

Soluciones relacionales y datos NoSQL

Artículo • 05/06/2023

Sugerencia

Este contenido es un extracto del libro electrónico “Architecting Cloud Native .NET Applications for Azure” (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Los datos relacionales y NoSQL son dos tipos de sistemas de base de datos que se implementan con frecuencia en las aplicaciones nativas de nube. Se diferencian en cómo se crean, la forma en la que almacenan datos y en cómo se accede a ellas. En esta sección, se abordarán los dos tipos. Más adelante en este capítulo, nos centraremos en una tecnología de base de datos emergente denominada *NewSQL*.

Las *bases de datos relacionales* son una tecnología predominante desde hace décadas. Demuestran madurez, se han probado y son ampliamente implementadas. Los productos de base de datos, las herramientas y la experiencia de la competencia son abundantes. Las bases de datos relacionales proporcionan un almacén de tablas de datos relacionadas. Estas tablas presentan un esquema fijo, usan SQL (Lenguaje de consulta estructurado) para administrar datos y admiten garantías ACID.

Las *bases de datos NoSQL* se refieren a los almacenes de datos no relacionales y de alto rendimiento. Destacan por su facilidad de uso, escalabilidad, resistencia y disponibilidad. En lugar de combinar tablas de datos normalizados, NoSQL almacena datos no estructurados o semiestructurados, a menudo en pares clave-valor o documentos JSON. Los base de datos no SQL no suelen ofrecen garantías ACID aparte del ámbito de una única partición de base de datos. Los servicios de gran volumen que requieren tiempos de respuesta por debajo de un segundo dan prioridad a los almacenes de datos NoSQL.

Es necesario insistir en la importancia del efecto de las tecnologías NoSQL en sistemas nativos de nube distribuidos. La proliferación de nuevas tecnologías de datos en este espacio ha alterado las soluciones que antes confiaban exclusivamente en bases de datos relacionales.

Las bases de datos NoSQL incluyen varios modelos diferentes para acceder a los datos y administrarlos, cada uno adecuado para casos de uso concretos. En la figura 5-9 se presentan cuatro modelos frecuentes.

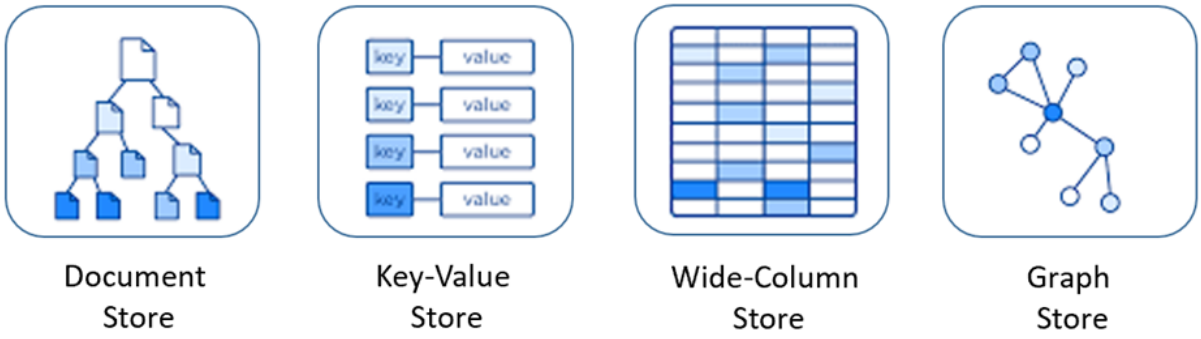


Figura 5-9: modelos de datos para bases de datos NoSQL

Modelo	Características
Almacén de documentos	Los datos y los metadatos se almacenan jerárquicamente en documentos basados en JSON dentro de la base de datos.
Almacén de clave-valor	La más sencilla de las bases de datos NoSQL, los datos se representan como una colección de pares clave-valor.
Almacén de columna ancha	Los datos relacionados se almacenan como un conjunto de pares clave-valor anidados dentro de una sola columna.
Almacén de grafos	Los datos se almacenan en una estructura de grafo como propiedades de nodo, borde y datos.

Teorema CAP

Para entender las diferencias entre estos tipos de bases de datos, piense en el teorema CAP, una serie de principios aplicados a los sistemas distribuidos que almacenan el estado. En la figura 5-10 se muestran las tres propiedades del teorema CAP.

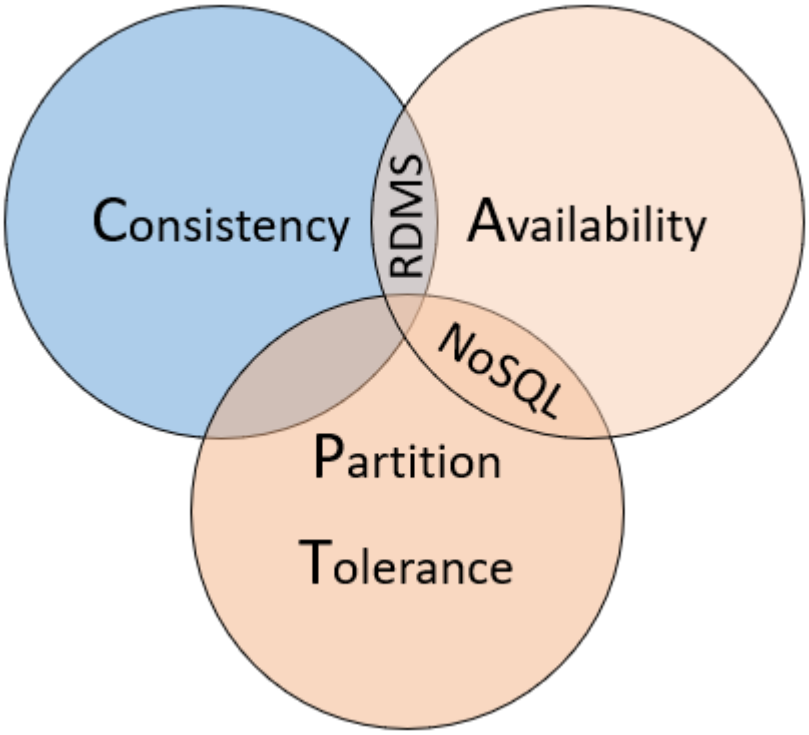


Figura 5-10. Teorema CAP

El teorema indica que los sistemas de datos distribuidos ofrecerán un equilibrio entre la coherencia, la disponibilidad y la tolerancia a particiones. Además, cualquier almacén de datos solo puede garantizar *dos* de las tres propiedades:

- *Coherencia* Cada nodo del clúster responde con los datos más recientes, incluso si el sistema debe bloquear la solicitud hasta que se actualicen todas las réplicas. Si consulta un «sistema coherente» para un artículo que se está actualizando en ese momento, tendrá que esperar hasta que todas las réplicas se actualicen correctamente. Sin embargo, recibirá los datos más actualizados.

- *La disponibilidad.* Cada nodo devuelve una respuesta inmediata, incluso si esa respuesta no son los datos más recientes. Si consulta un «sistema disponible» para un artículo que se está actualizando, obtendrá la mejor respuesta posible que el servicio puede ofrecer en ese momento.

- *Tolerancia a la partición.* Garantiza que el sistema sigue funcionando incluso si se produce un error en un nodo de datos replicados o se pierde la conectividad con otros nodos de datos replicados.

El teorema CAP explica la solución de compromiso que se adopta en la administración de la coherencia y disponibilidad durante una partición de red. Sin embargo, las soluciones de compromiso relacionadas con la coherencia y el rendimiento existen también sin una partición de red. El teorema CAP se suele ampliar a PACELC para explicar las soluciones de compromiso globalmente.

Las bases de datos relacionales suelen proporcionar coherencia y disponibilidad, pero no tolerancia a particiones. Normalmente se aprovisionan en un único servidor y se escalan verticalmente al agregar más recursos a la máquina.

Muchas bases de datos relacionales admiten características de replica integradas en las que se pueden hacer copias de la bases de datos principales en otras instancias de servidor secundarias. Las operaciones de escritura se hacen en la instancia principal y se replican en cada una de las secundarias. Tras un error, la instancia principal puede conmutar por error en una secundaria para proporcionar alta disponibilidad. Las instancias secundarias también se pueden usar operaciones de lectura distribuidas. Aunque las operaciones de escritura siempre van en contra de la réplica principal, las operaciones de lectura se pueden enrutar a cualquiera de las secundarias para reducir la carga del sistema.

Los datos se pueden particionar horizontalmente en varios nodos, como con particionamiento. Sin embargo, el particionamiento aumenta drásticamente la sobrecarga operativa al dividir los datos en muchas partes que no se pueden comunicar fácilmente. Administrarlo puede ser costoso y lento. Las características relacionales que incluyen combinaciones de tablas, transacciones e integridad referencial necesitan penalizaciones de rendimiento en las implementaciones particionadas.

Es posible optimizar la coherencia de la réplica y los objetivos de punto de recuperación al configurar si la réplica se produce sincrónica o asincrónicamente. Si las réplicas de datos perdieran conectividad de red en un clúster de bases de datos relacionales «altamente coherentes» o sincrónicas, no podría escribir en la base de datos. El sistema rechazaría la operación de escritura porque no podría replicar ese cambio en la otra réplica de datos. Cada réplica de datos tiene que actualizarse antes de que pueda completar la transacción.

Las bases de datos NoSQL suelen admitir alta disponibilidad y tolerancia a la partición. Se escalan horizontalmente, a menudo en servidores de productos básicos. Con este enfoque se proporciona una gran disponibilidad, tanto entre como en las regiones geográficas a un coste reducido. Puede crear particiones y replicar datos en estas máquinas o nodos, lo que proporciona redundancia y tolerancia a errores. La coherencia se suele ajustar por medio de protocolos de consenso o mecanismos de quorum. Se obtiene un mayor control para explorar soluciones de compromiso en las que se tiene que elegir entre una réplica sincrónica y una asincrónica en sistemas relacionales.

Si las réplicas de datos perdieran conectividad en un clúster de base de datos NoSQL de «alta disponibilidad», se podría completar una operación de escritura en la base de datos. El clúster de base de datos permitiría la operación de escritura y actualizaría cada réplica de datos a medida que estén disponibles. Las bases de datos NoSQL que

admiten varias réplicas grabables puede reforzar todavía más la alta disponibilidad evitando que sea necesaria una conmutación por error al optimizar el objetivo del tiempo de recuperación.

Las bases de datos NoSQL modernas suelen implementar funcionalidades de creación de particiones como una característica del diseño del sistema. La administración de particiones suele estar integrada en la base de datos y el enrutamiento se obtiene por medio de sugerencias de selección de ubicación, que se suelen llamar claves de partición. Mediante un modelo de datos flexible, las bases de datos NoSQL reducen la carga de administración de esquemas y mejoran la disponibilidad al implementar actualizaciones de aplicación que requieren modificaciones en el modelo de datos.

A menudo la alta disponibilidad y la escalabilidad masiva son más importantes para la empresa que las combinaciones de tablas relacionales y la integridad referencial. Los desarrolladores pueden implementar técnicas y patrones como Sagas, CQRS y mensajería asincrónica para adoptar la coherencia final.

> Hoy en día, conviene prestar atención al considerar las restricciones del teorema CAP. Un nuevo tipo de bases de datos, denominada NewSQL, ha surgido y amplía el motor de base de datos relacional para admitir la escalabilidad horizontal y el rendimiento escalable de los sistemas NoSQL.

Cuestiones sobre las diferencias entre los sistemas relacionales y NoSQL

En función de los requisitos de datos específicos, un microservicio nativo puede implementar un almacén de datos relacional, una base de datos NoSQL o ambos.

Plantéese un almacén de datos NoSQL cuando:	**Plantéese una base de datos relacional cuando:**
Tiene cargas de trabajo de gran volumen que requieren una latencia predecible a gran escala (por ejemplo, la latencia medida en milisegundos mientras hace millones de transacciones por segundo)	El volumen de carga de trabajo por lo general se ajusta a miles de transacciones por segundo
Los datos son dinámicos y cambian frecuentemente	Los datos están altamente estructurados y necesitan integridad referencial
Las relaciones pueden ser modelos de datos desnormalizados	Las relaciones se expresan por medio de combinaciones de tablas en modelos de datos normalizados

Plantéese un almacén de datos NoSQL cuando:	Plantéese una base de datos relacional cuando:
La recuperación de datos es sencilla y se expresa sin combinaciones de tabla	Maneja consultas e informes complejos
Los datos se suelen replicar entre zonas geográficas y se requiere un control más preciso sobre la coherencia, la disponibilidad y el rendimiento	Normalmente, los datos están centralizados o pueden replicarse regiones de forma asincrónica
La aplicación se implementará en hardware de productos básicos, como con nubes públicas	La aplicación se implementará en un hardware grande y de gama alta

En las secciones siguientes, se explorarán las opciones disponibles en la nube de Azure para almacenar y administrar los datos nativos de nube.

Base de datos como servicio

Para empezar, puede aprovisionar una máquina virtual de Azure e instalar las bases de datos que prefiere para cada servicio. Aunque tiene control absoluto sobre el entorno, renunciaría a muchas características integradas de la plataforma en la nube. También sería responsable de administrar la máquina virtual y la base de datos de cada servicio. Este enfoque podría convertirse rápidamente en una solución cara y lenta.

En su lugar, las aplicaciones nativas de nube priorizan servicios de datos expuestos como una base de datos como servicio (DBaaS). Totalmente administrado por un proveedor en la nube, estos servicios ofrecen seguridad, escalabilidad y supervisión integradas. En lugar de ser propietario del servicio, simplemente se consume como un servicio de respaldo. El proveedor opera el recurso a escala y asume la responsabilidad del rendimiento y del mantenimiento.

Pueden configurarse en las zonas de disponibilidad en la nube y las regiones para obtener una alta disponibilidad. Todos admiten la capacidad Just-In-Time y el modelo de pago por uso. Azure ofrece diferentes tipos de opciones de servicios de datos administrados, cada una con ventajas específicas.

En primer lugar, trataremos los servicios de base de datos como servicios relacionales disponibles en Azure. Verá que la base de datos de SQL Server, buque insignia de Microsoft, está disponible con varias opciones de código abierto. A continuación, abordaremos los servicios de datos NoSQL en Azure.

Bases de datos relacionales de Azure

Para los microservicios nativos de nube que requieren datos relacionales, Azure proporciona cuatro ofertas bases de datos relacionales administradas como servicio, como se muestra en la figura 5-11.

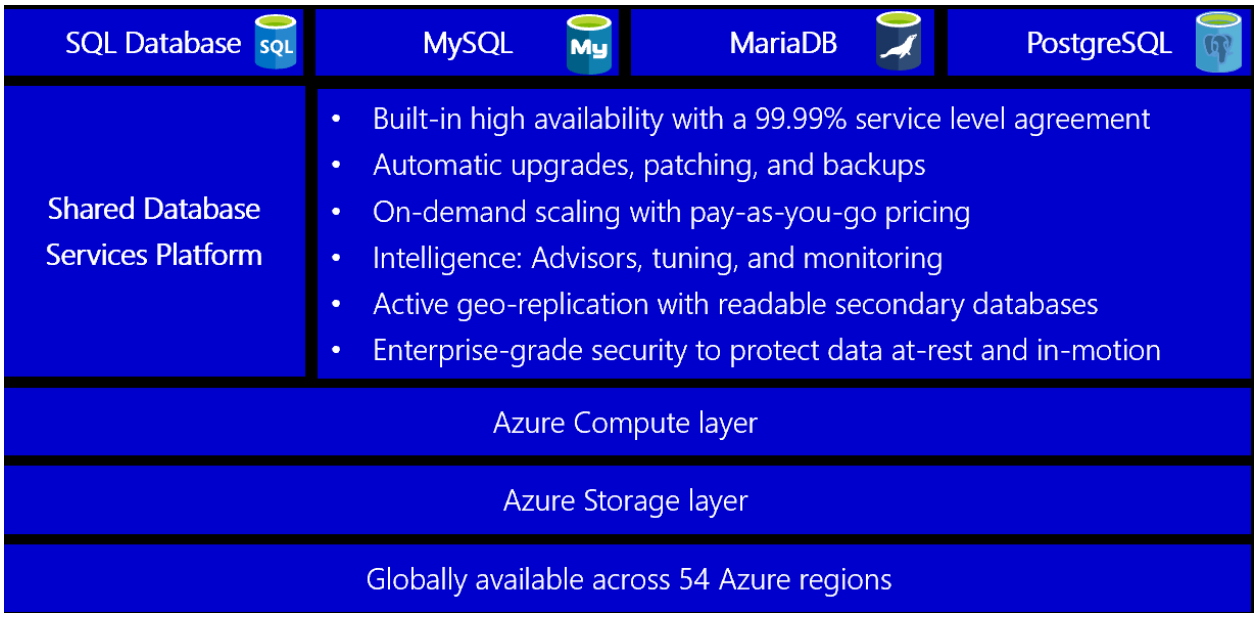


Figura 5-11. Bases de datos relacionales administradas disponibles con Azure

En la figura anterior se puede ver cómo cada una se encuentra en una infraestructura de base de datos como servicios común con funcionalidades clave sin coste adicional.

Estas funcionalidades son especialmente importantes para organizaciones que aprovisionan grandes cantidades de bases de datos, pero no tienen muchos servicios para administrarlas. Puede aprovisionar una base de datos de Azure en minutos al seleccionar la cantidad de núcleos de procesamiento, memoria y almacenamiento subyacente. Puede escalar la base de datos sobre la marcha y ajusta dinámicamente los recursos con poco o sin tiempo de inactividad.

Azure SQL Database

Los equipos de desarrollo que tienen experiencia en Microsoft SQL Server debería considerar Azure SQL Database. Es una base de datos como servicios relacional totalmente administrada que se basa en el Motor de base de datos de Microsoft SQL Server. El servicio comparte muchas características con la versión local de SQL Server y ejecuta la última versión estable del motor de base de datos de SQL Server.

Para usarlo con un microservicio nativo de nube, Azure SQL Database está disponible con tres opciones de implementación:

- Una Base de datos única representa una SQL Database totalmente administrada que se ejecuta en un servidor de Azure SQL Database en la nube de Azure. La base de datos se considera *independiente*, dado que no tiene dependencias de configuración en el servidor de bases de datos subyacente.

- Una instancia administrada es una instancia totalmente administrada del motor de base de datos de Microsoft SQL Server, que proporciona casi el 100 % de compatibilidad con un SQL Server local. Esta opción admite bases de datos más amplias, de hasta 35 TB, y se coloca en una Virtual Network de Azure para obtener mejor aislamiento.

- Azure SQL Database sin servidor es un nivel de proceso para una base de datos única que escala automáticamente en función de la demanda de la carga de trabajo. Solo se factura la cantidad de proceso utilizado por segundo. El servicio es adecuado para cargas de trabajo con patrones de uso intermitentes e impredecibles, intercalados con períodos de inactividad. Además, el nivel de proceso sin servidor pausa automáticamente las bases de datos durante períodos de inactividad para que se factura solo el almacenamiento. Se reactiva automáticamente al reanudarse la actividad.

Además de la pila tradicional de Microsoft SQL Server, Azure también dispone de versiones administradas de tres bases de datos de código abierto.

Bases de datos de código abierto de Azure

Las bases de datos relacionales de código abierto han alcanzado una gran popularidad para las aplicaciones nativas de nube. Cuentan con la prioridad que les dan muchas empresas frente a productos de bases de datos comerciales, en especial por razones económicas. A muchos equipos de desarrollo les interesa su flexibilidad, el desarrollo respaldado por la comunidad y su ecosistema de herramientas y extensiones. Las bases de datos de código abierto pueden implementarse en varios proveedores de nube, con lo que se minimiza el problema del «bloqueo del vendedor».

Los desarrolladores pueden hospedar de forma sencilla cualquier base de datos de código abierto en una máquina virtual de Azure. Al proporcionarle control absoluto, con este enfoque usted es el encargado de la administración, supervisión y mantenimiento de la base de datos y máquina virtual.

Sin embargo, Microsoft mantiene su compromiso de mantener Azure como una «plataforma abierta» al ofrecerle varias bases de datos de código abierto populares como servicios DBaaS *totalmente administrados*.

Azure Database for MySQL

MySQL es una base de datos relacional de código abierto y constituye la base de las aplicaciones integradas en la pila de software LAMP. Especialmente popular por las cargas de trabajo de *lectura intensiva*, la usan muchas compañías grandes, entre las que se incluyen Facebook, Twitter y YouTube. La community edition está disponible de forma gratuita, mientras que para la enterprise edition es necesario comprar una licencia. Creado en 1995, Sun Microsystems compró el producto en 2008. Oracle adquirió Sun y MySQL en 2010.

Azure Database for MySQL es un servicio de base de datos relacional administrado que se basa en el motor de código abierto MySQL. Usa la edición MySQL Community. El servidor de Azure MySQL es el punto administrativo del servicio. Es el mismo motor de servidor de MySQL que se usa en las implementaciones locales. El motor puede crear una base de datos única por servidor o varias bases de datos por servidor que comparten recursos. Puede seguir administrando los datos con las mismas herramientas de código abierto sin tener que adquirir aptitudes nuevas o administrar máquinas virtuales.

Azure Database for MariaDB

El servidor MariaDB Server es otro servidor de bases de datos de código abierto ampliamente usado. Se creó como *bifurcación* de MySQL cuando Oracle adquirió Sun Microsystems, propietario de MySQL. Se pretendía garantizar que MariaDB siguiera siendo de código abierto. Dado que MariaDB es una bifurcación de MySQL, las definiciones de los datos y las tablas son compatibles. Por su parte, los protocolos de cliente, las estructuras y las API están estrechamente unidas.

MariaDB cuenta con una comunidad sólida y la usan muchas grandes empresas. Mientras Oracle sigue manteniendo, mejorando y admitiendo MySQL, la fundación MariaDB administra MariaDB, lo que permite contribuciones públicas al producto y documentación.

Azure Database for MariaDB es una base de datos relacional totalmente administrada como servicio en la nube de Azure. Este servicio se basa en el motor de servidor de MariaDB Community Edition. Puede controlar cargas de trabajo críticas con un rendimiento predecible y escalabilidad dinámica.

Azure Database for PostgreSQL

PostgreSQL es una base de datos relacional de código abierto con más de treinta años de desarrollo activo. PostgreSQL debe su gran reputación a su confiabilidad e

integridad de datos. Tiene muchas funcionalidades, es compatible con SQL y se considera más eficaz que MySQL, especialmente para cargas de trabajo con consultas complejas y escrituras intensivas. Muchas grandes empresas, entre las que se incluyen Apple, Red Hat y Fujitsu, han creado productos con PostgreSQL.

Azure Database for PostgreSQL es un servicio de base de datos relacional totalmente administrado basado en el motor de base de datos de código abierto Postgres. El servicio admite muchas plataformas de desarrollo, como C++, Java, Python, Node, C# y PHP. Se pueden migrar las bases de datos de PostgreSQL a este servicio, con la herramienta de interfaz de la línea de comandos o Azure Data Migration Service.

Azure Database for PostgreSQL está disponible con dos opciones de implementación:

- La opción de implementar un único servidor en un punto administrativo central para varias bases de datos en las que se pueden implementar muchas bases de datos. Los precios se calculan por servidor en función de los núcleos y el almacenamiento.
- La opción Hiperescala (Citus) funciona con la tecnología Citus Data. Permite alto rendimiento al *escalar horizontalmente* una base de datos única en cientos de nodos para ofrecer un rendimiento y una escala rápidos. De este modo, el motor puede ajustarse a más datos en la memoria, paralelizar consultas en cientos de nodos e indexar los datos con mayor rapidez.

Datos NoSQL en Azure

Cosmos DB es un servicio de base de datos NoSQL distribuida globalmente y totalmente administrada en la nube de Azure. Muchas grandes empresas en todo el mundo la han adoptado, entre las que se incluyen Coca-Cola, Skype, ExxonMobil y Liberty Mutual.

Si los servicios requieren una respuesta rápida desde cualquier punto del globo, alta disponibilidad o escalabilidad elástica, Cosmos DB es una excelente opción. En la figura 5-12 se muestra Cosmos DB.

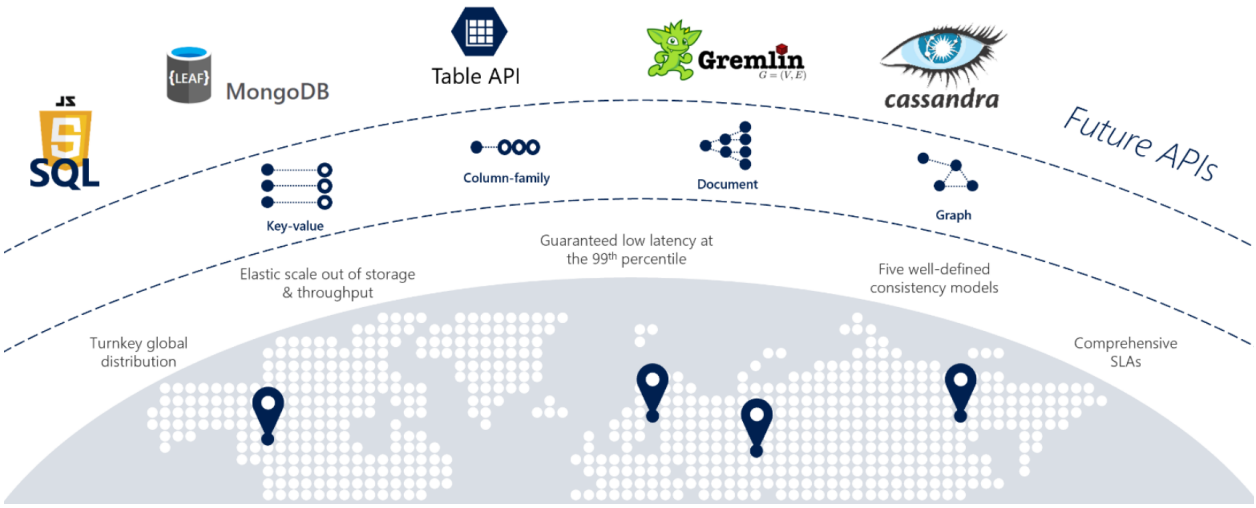


Figura 5-12: Introducción a Azure Cosmos DB

En la figura anterior se presentan muchas funcionalidades integradas nativas de nube disponibles en Cosmos DB. En este apartado las veremos con más detalle.

Compatibilidad global

Las aplicaciones nativas de nube suelen tener una audiencia global y requieren una escala global.

Puede distribuir bases de datos de Cosmos en regiones o en todo el mundo, acercando los datos a sus usuarios, mejorando el tiempo de respuesta y reduciendo la latencia. Puede agregar o eliminar una base de datos de una región sin pausar o volver a implementar los servicios. En segundo plano, Cosmos DB replica de forma transparente los datos a cada una de las regiones configuradas.

Cosmos DB admite la agrupación en clústeres activo/activo a un nivel global, lo que le permite configurar una de las regiones de la base de datos para admitir *escrituras y lecturas*.

El protocolo de escribir en varias regiones es una característica importante de Cosmos DB que habilita las características siguientes:

- Escalabilidad de escritura y lectura elásticas ilimitada.
- 99,999 % de disponibilidad de lectura y escritura en todo el mundo.
- Garantía de lecturas y escrituras atendidas en menos de 10 milisegundos en el percentil 99.

Con las API de hospedaje múltiple de Cosmos DB, el microservicio conoce automáticamente la región de Azure más próxima y le envía solicitudes. Cosmos DB

identifica la región más cercana sin hacer cambios de configuración. Si una región deja de estar disponible, la característica de hospedaje múltiple solicitará automáticamente la ruta a la siguiente región disponible más cercana.

Compatibilidad con varios modelos

Al transportar aplicaciones monolíticas a una arquitectura nativa de nube, los equipos de desarrollo a veces tienen que migrar almacenes de datos NoSQL de código abierto. Mediante Cosmos DB, no se pierde la inversión en estos almacenes de datos NoSQL con su plataforma de datos de *varios modelos*. En la tabla siguiente se muestran las API de compatibilidad NoSQL admitidas.

Proveedor	Descripción
NoSQL API	La API para NoSQL almacena datos en formato de documento
API de Mongo DB	Admite las API de Mongo DB y los documentos JSON
API de Gremlin	Admite la API de Gremlin con nodos basados en gráficos y representaciones de datos perimetrales
Cassandra API	Admite la API de Casandra para representar datos de columnas anchas
Table API	Admite Azure Table Storage con mejoras premium
API PostgreSQL	Servicio administrado para ejecutar PostgreSQL a cualquier escala

Los equipos de desarrollo puede migrar bases de datos existentes de Mongo, Gremlin o Cassandra en Cosmos DB con mínimas modificaciones en los datos o el código. En el caso de las nuevas aplicaciones, los equipos de desarrollo pueden decidirse por una de las opciones de código abierto o el modelo de API de SQL integrado.

> De forma interna, Cosmos almacena los datos en un formato de estructura simple formado por tipos de datos primitivos. En cada solicitud, el motor de base de datos traduce los datos primitivos en la representación del modelo que ha seleccionado.

En la tabla anterior, anote la opción Table API. Esta API es una evolución de Table Storage de Azure. Las dos comparten el mismo modelo de tabla subyacente, pero la Table API de Cosmos DB agrega mejoras premium que no incluye la API de Azure Storage. En la tabla siguiente se comparan las características.

Característica	Azure Table Storage	Azure Cosmos DB

Característica	Azure Table Storage	Azure Cosmos DB
Latencia	Rápido	La latencia de milisegundos de un solo dígito para lecturas y escrituras en cualquier parte del mundo
Throughput	Límite de 20 000 operaciones por tabla	Operaciones ilimitadas por tabla
Distribución global	Región única con una única región de lectura secundaria	Distribuciones de llave en mano a todas las regiones con conmutación por error automática
Indización	Solo está disponible para las propiedades de partición y clave de fila	Indexación automática de todas las propiedades
Precios	Optimizados por cargas de trabajo inactivas (bajo rendimiento: proporción de almacenamiento)	Optimizados para cargas de trabajo activas (alto rendimiento: proporción de almacenamiento)

Los microservicios que consumen almacenamiento de las tablas de Azure pueden migrar a Table API de Cosmos DB sin problemas. No se requiere ningún cambio de código.

Coherencia optimizable

Anteriormente, en el apartado *Soluciones relacionales y datos NoSQL* se explica el asunto de la *coherencia de los datos*. La coherencia de los datos hace referencia a la *integridad* de los datos. Los servicios nativos de nube con datos distribuidos dependen de la réplica y deben adoptar una solución de compromiso y elegir entre coherencia, la disponibilidad y la latencia de lectura.

La mayoría de las bases de datos distribuidas permiten a los desarrolladores elegir entre dos modelos de coherencia: coherencia fuerte y coherencia final. La *coherencia fuerte* es el estándar de oro de la programación de datos. Garantiza que la consulta siempre devolverá los datos más actuales, incluso si el sistema deben incurrir en latencia esperando una actualización que se replique en todas las copias de bases de datos. Una base de datos configurada para la *coherencia final* devolverá datos inmediatamente, incluso si los datos no son la copia más actual. Esta última opción permite una disponibilidad más alta, una mayor escala y un rendimiento aumentado.

Azure Cosmos DB ofrece cinco modelos de coherencia bien definidos que se muestran en la figura 5-13.

Figura 5-13: niveles de coherencia de Cosmos DB

Con estas opciones, puede tomar decisiones precisas y soluciones de compromiso granulares para la coherencia, la disponibilidad y el rendimiento de los datos. En la siguiente tabla se presentan los niveles.

Nivel de coherencia	Descripción
Ocasional	No existen garantías de ordenación para las operaciones de lectura. Las réplicas convergirán en última instancia.
Prefijo constante	Las operaciones de lectura siguen siendo finales, pero se devuelven los datos en el mismo de orden en el que se escriben.
Sesión	Garantiza las operaciones de la lectura y la escritura de cualquier dato durante la sesión actual. Es el nivel de coherencia predeterminada.
De obsolescencia entrelazada	Lee las operaciones de escritura finales por el intervalo que especifica.
Alta	Se garantiza que las operaciones de las lecturas devolverán la versión confirmada más reciente de un elemento. Un cliente nunca ve una operación de lectura no confirmada ni parcial.

En el artículo Getting Behind the 9-Ball: Cosmos DB Consistency Levels Explained, (Introducción a la 9-Ball: explicación de los niveles de coherencia de la base de datos de Cosmos DB), Jeremy Likness, administrado de programas de Microsoft, ofrece una explicación excelente de los cinco modelos.

Creación de particiones

Azure Cosmos DB adopta la creación de particiones automáticamente para escalar una base de datos para satisfacer las necesidades de rendimiento de los servicios nativos de nube.

Los datos de Cosmos DB se administran al crear bases de datos, contenedores y elementos.

Los contenedores residen en una base de datos de Cosmos DB y representan una agrupación de elementos independiente al esquema. Los elementos son los datos que agrega al contenedor. Están representados como documentos, filas, nodos o bordes. Todos los elementos que se agregan a un contenedor se indexan automáticamente.

Para obtener la partición del contenedor, los elementos se dividen en subconjuntos distintos denominados particiones lógicas. Las particiones lógicas se rellenan en función del valor de una clave de partición que está asociada con cada elemento de un contenedor. En la figura 5-14 aparecen dos contenedores cada uno con una partición lógica basada en un valor de clave de partición.

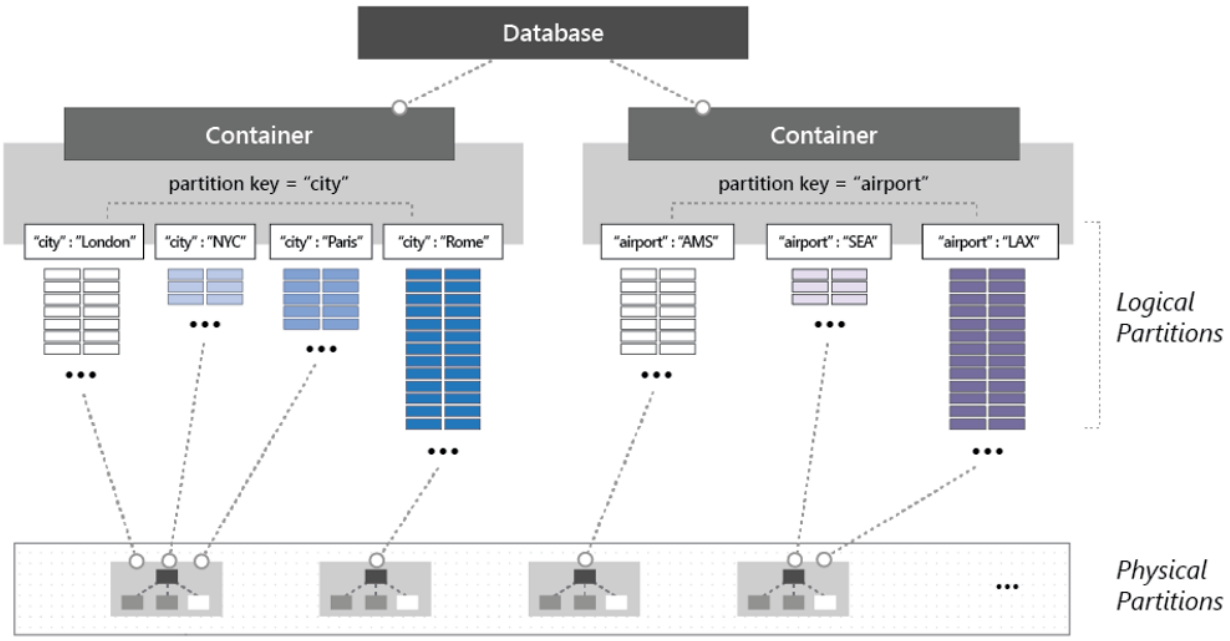


Figura 5-14: mecanismos de partición de Cosmos DB

En la anterior figura, preste atención a cómo cada elemento incluye una clave de partición de «ciudad» o «aeropuerto». La clave determina la partición lógica del elemento. Los elementos con un código de ciudad se asignan al contenedor de la izquierda y los que tienen un código de aeropuerto, al contenedor de la derecha. Al combinar el valor de clave de partición con el valor de id. se crea un índice de elemento, que identifica de forma única el elemento.

De forma interna, Cosmos DB administra automáticamente la ubicación de las particiones lógicas en particiones físicas para satisfacer las necesidades de escalabilidad y rendimiento del contenedor. Dato que los requisitos de almacenamiento y rendimiento aumentan, Azure Cosmos DB redistribuyen las particiones lógicas en un mayor de número de servidores. Cosmos DB administra las operaciones de redistribución y se invocan sin interrupciones ni tiempo de inactividad.

Bases de datos NewSQL

NewSQL es una tecnología de bases de datos emergente que combina la escalabilidad distribuida de NoSQL con las garantías ACID de una base de datos relacional. Las bases de datos de NewSQL son importantes para sistemas empresariales que tienen que procesar grandes volúmenes de datos, en entornos distribuidos, con compatibilidad transaccional completa y con ACID. Una base de datos NoSQL puede proporcionar escalabilidad masiva, pero no garantiza la coherencia de datos. Los problemas intermitentes de los datos incoherentes pueden suponer una carga en el equipo de desarrollo. Los desarrolladores deben crear medidas de seguridad en el código de microservicios para administrar problemas que causan los datos incoherentes.

La Cloud Native Computing Foundation (CNCF) (Fundación de Informática Nativa de Nube) cuenta con varios proyectos de bases de datos NewSQL.

Project	**Características**
CockroachDB	Una base de datos relacional compatible con ACID que se escala de forma universal. Si se agrega un nodo nuevo a un clúster, CockroachDB equilibrará los datos entre instancias y zonas geográficas. Crea, administra y distribuye réplicas para garantizar la confiabilidad. Es de código abierto y gratis.
TiDB	Una base de datos de código abierto que admite cargas de trabajo de procesamiento analítico y transaccional híbrido (HTAP). Es compatible con MySQL y cuenta con escalabilidad horizontal, coherencia fuerte y alta disponibilidad. TiDB se comporta como servidor MySQL. Es posible usar las bibliotecas de cliente MySQL existentes, sin hacer cambios radicales de código en la aplicación.
YugabyteDB	Una base de datos SQL distribuida, de código abierto y alto rendimiento. Admite latencia de consulta baja, resistencia frente a errores y distribución de datos global. YugabyteDB es compatible con PostgreSQL y controla las cargas de trabajo de RDBMS de escalabilidad horizontal y OLTP a escala de Internet. El producto también admite NoSQL y es compatible con Cassandra.
Vitess	Vitess es una solución de bases de datos para la implementación, el escalado y la administración de clústeres amplios de las instancias MySQL. Se puede ejecutar en una arquitectura de nube privada o pública. Vitess combina y amplía muchas características importantes de MySQL y es compatible con particionamiento horizontal y verticales. Creado por YouTube, Vitess ha atendiendo todo el tráfico de base de datos de YouTube desde 2011.

Los proyectos de código abierto de la figura anterior están disponibles en la Cloud Native Computing Foundation. Tres de las ofertas son productos completos de base de datos, que incluyen compatibilidad con .NET. La otra, Vitess, es un sistema de agrupación en clústeres de base de datos que escala horizontalmente grandes clústeres de instancias de MySQL.

Las bases de datos NewSQL tienen como uno de sus objetivos clave de diseño operar de forma nativa en Kubernetes, aprovechando las ventajas de la resistencia y escalabilidad de la plataforma.

Las bases de datos NewSQL están pensadas para desarrollarse en entornos de nube efímeros en los que se pueden reiniciar máquinas virtuales subyacentes o se pueden programar en un momento dato sin previo aviso. Las bases de datos están pensadas para sobrevivir a errores de nodos sin perder datos ni tiempo de inactividad. CockroachDB, por ejemplo, puede sobrevivir a una pérdida de máquina al mantener tres réplicas coherentes de cualquier dato en los nodos en un clúster.

Kubernetes usa una *construcción de servicios* para permitir que un cliente aborde un grupo de procesos NewSQL idénticos desde una única entrada DNS. Al desacoplar las instancias de bases de datos desde la dirección del servicio al que se asocia, se puede escalar sin interrumpir las instancias de aplicación existentes. Al enviar una solicitud a cualquier servicio en un momento dato, siempre se obtendrá el mismo resultado.

En este escenario, todas las instancias de bases de datos son iguales. No hay relaciones principales ni secundarias. Mediante las técnicas como la *réplica de consenso* que se encuentran en CockroachDB, cualquier modo de base de datos puede controlar cualquier solicitud. Si el nodo que recibe una solicitud de carga equilibrada tiene los datos que necesita localmente, responde inmediatamente. De lo contrario, el nodo se convierte en una puerta de enlace y reenvía la solicitud a los nodos adecuados para obtener la respuesta correcta. Desde la perspectiva del cliente, cada nodo de base de datos es igual: aparecen como una base de datos *lógica* única con las garantías de coherencia de un sistema de una única máquina, a pesar de tener docenas o incluso cientos de nodos activos en segundo plano.

Consulte el artículo DASH: Four Properties of Kubernetes-Native Databases, (DASH: cuatro propiedades de las bases de datos nativas de Kubernetes), para obtener una visión detallada de los mecanismos detrás de las bases de datos NewSQL.

Migración de datos a la nube

Una de las tareas más lentas es migrar datos de una plataforma de datos a otra. Con Azure Data Migration Service se pueden acelerar estas tareas. Puede migrar datos de varios orígenes de bases de datos externos a las plataformas de Azure Data con tiempo de inactividad mínimo. Entre las plataformas de destino se incluyen los siguientes servicios:

- Azure SQL Database
- Azure Database for MySQL

- Azure Database for MariaDB
- Azure Database for PostgreSQL
- Azure Cosmos DB

El servicio proporciona recomendaciones para servirle de orientación por los cambios necesarios para ejecutar una migración pequeña o grande.

Anterior Siguiente

Almacenamiento en caché en una aplicación nativa de nube

22/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Las ventajas del almacenamiento en caché son bien comprendidas. La técnica funciona copiando temporalmente los datos a los que se accede con frecuencia desde un almacén de datos back-end a *un almacenamiento rápido* que se encuentra más cerca de la aplicación. El almacenamiento en caché se implementa a menudo donde...

- Los datos permanecen relativamente estáticos.
- El acceso a datos es lento, especialmente en comparación con la velocidad de la memoria caché.
- Los datos están sujetos a altos niveles de contención.

¿Por qué?

Como se describe en la guía de almacenamiento en caché de Microsoft, el almacenamiento en caché puede aumentar el rendimiento, la escalabilidad y la disponibilidad de los microservicios individuales y el sistema en su conjunto. Reduce la latencia y la contención de controlar grandes volúmenes de solicitudes simultáneas en un almacén de datos. A medida que aumenta el volumen de datos y el número de usuarios, mayor será la ventaja del almacenamiento en caché.

El almacenamiento en caché es más eficaz cuando un cliente lee repetidamente los datos inmutables o que cambian con poca frecuencia. Algunos ejemplos incluyen información de referencia, como información de productos y precios, o recursos estáticos compartidos que son costosos de construir.

Aunque los microservicios deben no tener estado, una caché distribuida puede admitir el acceso simultáneo a los datos de estado de sesión cuando sea absolutamente necesario.

Considere también la posibilidad de almacenar en caché para evitar cálculos repetitivos. Si una operación transforma datos o realiza un cálculo complicado, almacene en caché el resultado de las solicitudes posteriores.

Arquitectura de almacenamiento en caché

Las aplicaciones nativas en la nube suelen implementar una arquitectura de almacenamiento en caché distribuida. La memoria caché se hospeda como un servicio de respaldo basado en la nube, independiente de los microservicios. En la figura 5-15 se muestra la arquitectura.

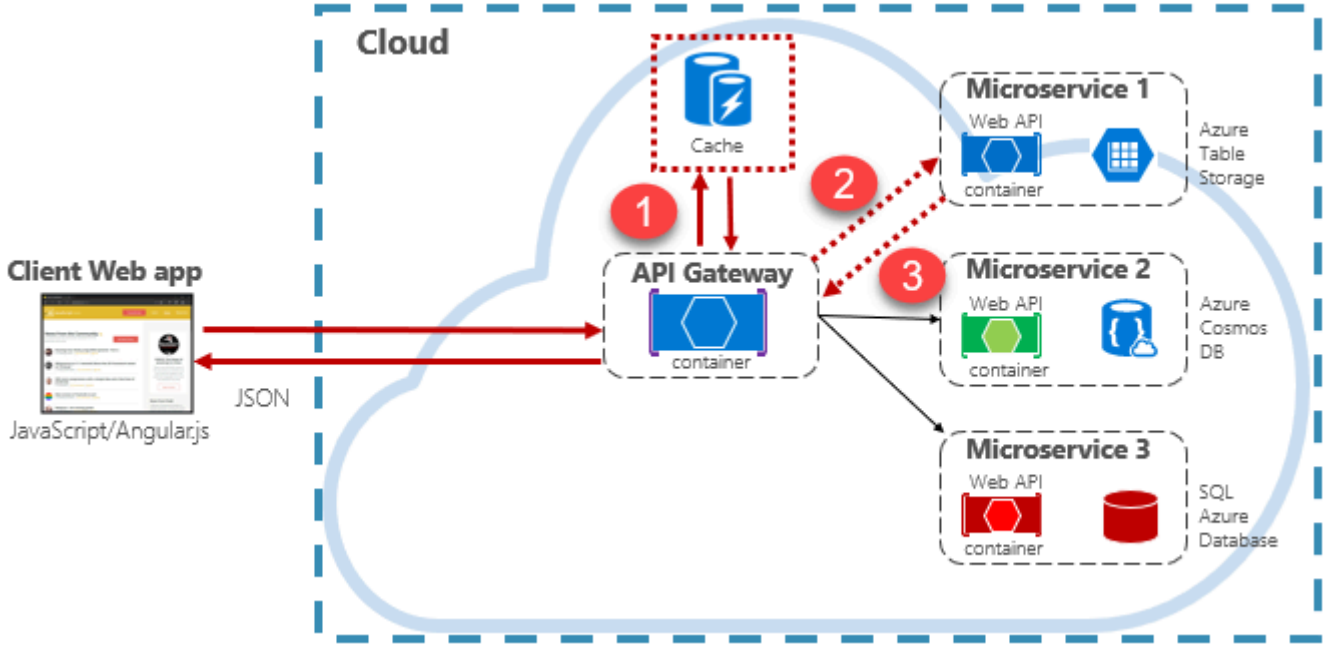


Figura 5-15: Almacenamiento en caché en una aplicación nativa en la nube

En la ilustración anterior, observe cómo la memoria caché es independiente y compartida por los microservicios. En este escenario, la puerta de enlace de API invoca la memoria caché. Como se describe en el capítulo 4, la puerta de enlace actúa como front-end para todas las solicitudes entrantes. La caché distribuida aumenta la capacidad de respuesta del sistema devolviendo los datos almacenados en caché siempre que sea posible. Además, separar la memoria caché de los servicios permite que la memoria caché se escale vertical o horizontalmente de forma independiente para satisfacer mayores demandas de tráfico.

En la ilustración anterior se presenta un patrón de almacenamiento en caché común conocido como patrón cache-aside. Para una solicitud entrante, primero debe consultar la memoria caché (paso 1) para obtener una respuesta. Si se encuentra, los datos se devuelven inmediatamente. Si los datos no existen en la memoria caché (conocido como error de caché), se recupera de una base de datos local en un servicio de bajada (paso 2). A continuación, se escribe en la memoria caché para las solicitudes futuras (paso 3) y se devuelve al autor de la llamada. Se debe tener cuidado para expulsar periódicamente los datos almacenados en caché para que el sistema permanezca oportuno y coherente.

A medida que crece una caché compartida, podría resultar beneficioso crear particiones de sus datos en varios nodos. Si lo hace, puede ayudar a minimizar la contención y mejorar la escalabilidad. Muchos servicios de almacenamiento en caché admiten la capacidad de agregar y quitar dinámicamente nodos y reequilibrar datos entre particiones. Este enfoque suele implicar la agrupación en clústeres. La agrupación en clústeres expone una colección de nodos federados como una caché única sin problemas. Sin embargo, internamente, los datos se dispersan entre los nodos siguiendo una estrategia de distribución predefinida que equilibra la carga uniformemente.

Caché de Azure para Redis

Azure Cache for Redis es un servicio seguro de agente de mensajería y almacenamiento en caché de datos, totalmente administrado por Microsoft. Consumido como oferta de plataforma como servicio (PaaS), proporciona un alto rendimiento y acceso de baja latencia a los datos. El servicio es accesible para cualquier aplicación dentro o fuera de Azure.

El servicio Azure Cache for Redis administra el acceso a los servidores redis de código abierto hospedados en centros de datos de Azure. El servicio actúa como una fachada que proporciona administración, control de acceso y seguridad. El servicio admite de forma nativa un amplio conjunto de estructuras de datos, incluidas cadenas, hashes, listas y conjuntos. Si la aplicación ya usa Redis, funcionará as-is con Azure Cache for Redis.

Azure Cache for Redis es más que un servidor de caché simple. Puede admitir una serie de escenarios para mejorar una arquitectura de microservicios:

- Un almacén de datos en memoria
- Una base de datos no relacional distribuida
- Un agente de mensajes
- Un servidor de configuración o detección

En escenarios avanzados, se puede conservar una copia de los datos almacenados en caché en el disco. Si un evento catastrófico deshabilita las memorias caché principal y de réplica, la memoria caché se reconstruye a partir de la instantánea más reciente.

Azure Redis Cache está disponible en una serie de configuraciones predefinidas y planes de tarifa. El nivel Premium incluye muchas características de nivel empresarial, como la agrupación en clústeres, la persistencia de datos, la replicación geográfica y el aislamiento de red virtual.

Siguiente anterior

Elasticsearch en una aplicación nativa de nube

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico "Architecting Cloud Native .NET Applications for Azure" (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Elasticsearch es un sistema de búsqueda y análisis distribuido que permite funcionalidades de búsqueda complejas en diversos tipos de datos. Es código abierto ampliamente conocido. Considere cómo las siguientes empresas integran Elasticsearch en su aplicación:

- Wikipedia para la búsqueda de texto completo e incremental (buscar mientras escribe).
- GitHub para indexar y exponer más de 8 millones de repositorios de código.
- Docker para que su biblioteca de contenedores sea reconocible.

Elasticsearch se basa en el motor de búsqueda de texto completo de Apache Lucene. Lucene proporciona indexación y consulta de documentos de alto rendimiento. Los datos se indexan con un esquema de indexación invertido: en lugar de asignar páginas a palabras clave, asigna palabras clave a páginas igual que un glosario al final de un libro. Lucene presenta eficaces funcionalidades de sintaxis de consulta y puede consultar datos por:

- Término (una palabra completa)
- Prefijo (comienza con una palabra)

- Carácter comodín (mediante filtros "*" o "?")
- Frase (una secuencia de texto en un documento)
- Valor booleano (búsquedas complejas que combinan consultas)

Mientras que Lucene proporciona fontanería de bajo nivel para la búsqueda, Elasticsearch proporciona el servidor que se asienta sobre Lucene. Elasticsearch agrega funcionalidad de nivel superior para simplificar el trabajo de Lucene, incluida una API RESTful para acceder a la funcionalidad de indexación y búsqueda de Lucene. También proporciona una infraestructura distribuida capaz de escalabilidad masiva, tolerancia a errores y alta disponibilidad.

En el caso de aplicaciones nativas de nube más grandes con requisitos de búsqueda complejos, Elasticsearch está disponible como servicio administrado en Azure. Microsoft Azure Marketplace presenta plantillas preconfiguradas que los desarrolladores pueden usar para implementar un clúster de Elasticsearch en Azure.

Desde Microsoft Azure Marketplace, los desarrolladores pueden usar plantillas preconfiguradas creadas para implementar rápidamente un clúster de Elasticsearch en Azure. Con la oferta administrada por Azure, puede implementar hasta 50 nodos de datos, 20 nodos de coordinación y tres nodos maestros dedicados.

Resumen

En este capítulo se ha presentado una visión detallada de los datos en sistemas nativos de la nube. Comenzamos contrastando el almacenamiento de datos en aplicaciones monolíticas con patrones de almacenamiento de datos en sistemas nativos de nube. Hemos examinado los patrones de datos implementados en sistemas nativos de nube, incluidas las consultas entre servicios, las transacciones distribuidas y los patrones para tratar con sistemas de gran volumen. Hemos comparado los datos SQL y NoSQL. Hemos examinado las opciones de almacenamiento de datos disponibles en Azure que incluyen opciones centradas en Microsoft y de código abierto. Por último, hemos analizado el almacenamiento en caché y Elasticsearch en una aplicación nativa de nube.

Referencias

- Patrón de segregación de responsabilidades de comandos y consultas (CQRS).
- Patrón Event Sourcing
- ¿Por qué no es tolerante a particiones RDBMS en el teorema CAP y por qué está disponible?

- Materialized View
- Todo lo que realmente necesita saber sobre las bases de datos de código abierto
- Patrón de transacción de compensación
- Patrón Saga
- Patrones Saga | Implementación de transacciones empresariales mediante microservicios
- Patrón de transacción de compensación
- Detrás de la bola 9: explicación de los niveles de coherencia de Cosmos DB
- En bases de datos RDBMS, NoSQL y NewSQL. Entrevista con John Ryan
- SQL frente a NoSQL frente a NewSQL: comparación completa
- DASH: cuatro propiedades de bases de datos nativas de Kubernetes
- CockroachDB
- TiDB
- YugabyteDB
- Vitess
- Elasticsearch: The Definitive Guide (Elasticsearch: la guía definitiva).
- Introducción a Apache Lucene

Resistencia nativa de la nube

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

La resistencia es la capacidad de su sistema de reaccionar ante errores y seguir siendo funcional. No se trata de evitar errores, sino de aceptar errores y construir los servicios nativos de la nube para responder a él. Quiere volver a un estado de funcionamiento completo lo antes posible.

A diferencia de las aplicaciones monolíticas tradicionales, donde todo se ejecuta juntos en un único proceso, los sistemas nativos de la nube adoptan una arquitectura distribuida, como se muestra en la figura 6-1:

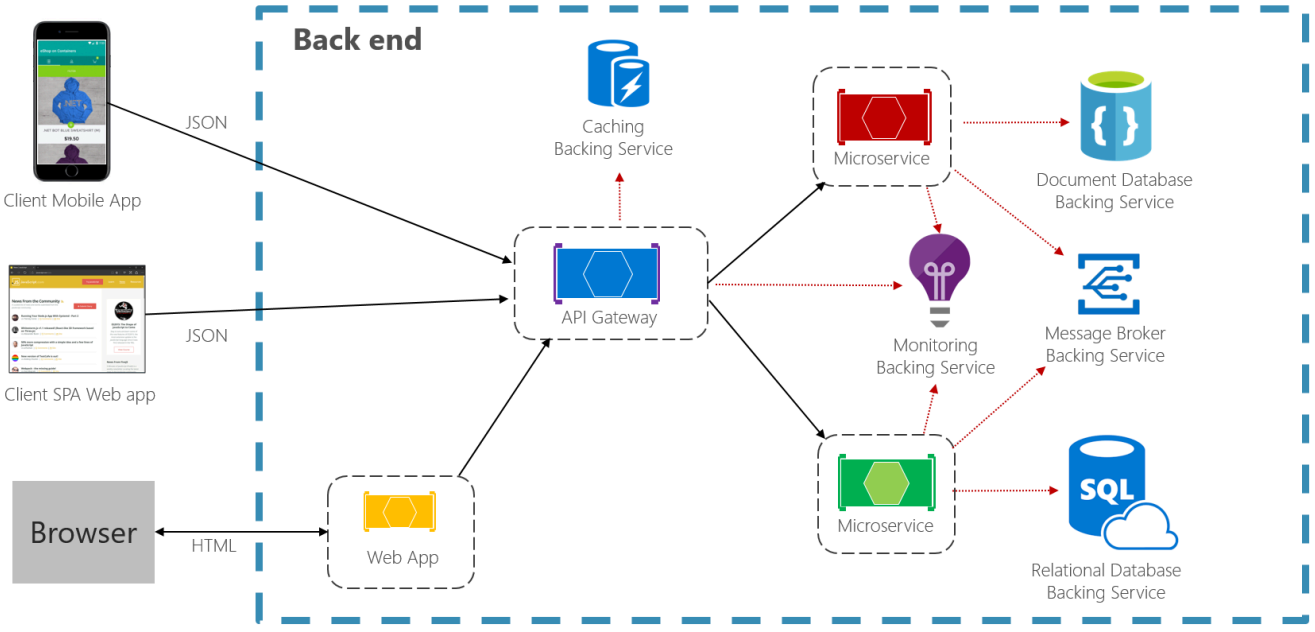


Figura 6-1. Entorno nativo de nube distribuido

En la ilustración anterior, cada microservicio y servicio de respaldo basado en la nube se ejecutan en un proceso independiente, en toda la infraestructura del servidor, que se comunica a través de llamadas basadas en red.

En este entorno, un servicio debe ser sensible a muchos desafíos diferentes:

- Latencia de red inesperada: el tiempo para que una solicitud de servicio viaje al receptor y de vuelta.
- Errores transitorios : errores de conectividad de red de corta duración.
- Bloqueo por una operación sincrónica de larga duración.
- Un proceso anfitrión que ha fallado y está siendo reiniciado o trasladado.
- Microservicio sobrecargado que no puede responder durante un breve tiempo.
- Una operación de orquestador en marcha como una actualización gradual o mover un servicio de un nodo a otro.
- Errores de hardware.

Las plataformas en la nube pueden detectar y mitigar muchos de estos problemas de infraestructura. Puede reiniciar, escalar horizontalmente e incluso redistribuir el servicio a otro nodo. Sin embargo, para aprovechar al máximo esta protección integrada, debe diseñar los servicios para reaccionar a él y prosperar en este entorno dinámico.

En las secciones siguientes, exploraremos las técnicas defensivas que el servicio y los recursos administrados en la nube pueden aprovechar para minimizar el tiempo de inactividad y la interrupción.

Siguiente anterior

Patrones de resistencia de aplicaciones

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

La primera línea de defensa es la resistencia de la aplicación.

Aunque podría invertir mucho tiempo escribiendo su propio marco de resistencia, estos productos ya existen. Polly es una biblioteca completa de resistencia de .NET y control de errores transitorios que permite a los desarrolladores expresar directivas de resistencia de forma fluida y segura para subprocesos. Polly tiene como destino las aplicaciones compiladas con .NET Framework o .NET 7. En la tabla siguiente se describen las características de resistencia, denominadas `policies`, disponibles en la biblioteca polly. Se pueden aplicar individualmente o agrupar.

Expandir tabla

Política	Experiencia
Volver a intentar	Configura las operaciones de reintento en las operaciones designadas.
Disyuntor	Bloquea las operaciones solicitadas durante un período predefinido cuando los errores superan un umbral configurado
Tiempo de espera	Coloca el límite durante el cual un autor de la llamada puede esperar una respuesta.
Compartimentado	Restringe las acciones a un grupo de recursos de tamaño fijo para evitar que las llamadas con errores saturen un recurso.

Política	Experiencia
Cache	Almacena las respuestas automáticamente.
Alternativa	Define el comportamiento estructurado tras un error.

Observe cómo, en la ilustración anterior, las directivas de resistencia se aplican a los mensajes de solicitud, tanto si proceden de un cliente externo como de un servicio back-end. El objetivo es compensar la solicitud de un servicio que podría ser momentáneamente no disponible. Estas interrupciones de corta duración suelen manifestarse con los códigos de estado HTTP que se muestran en la tabla siguiente.

Expandir tabla

Código de estado HTTP	Causa
404	No encontrado
408	Tiempo de espera de solicitud
429	Demasiadas solicitudes (lo más probable es que se haya limitado)
502	Puerta de enlace incorrecta
503	Servicio no disponible
504	Tiempo de espera de la puerta de enlace

Pregunta: ¿Reintentaría un código de estado HTTP de 403 - Prohibido? No. Aquí, el sistema funciona correctamente, pero informa al autor de la llamada de que no están autorizados para realizar la operación solicitada. Se debe tener cuidado para reintentar solo las operaciones causadas por errores.

Como se recomienda en el capítulo 1, los desarrolladores de Microsoft que construyen aplicaciones nativas de nube deben tener como destino la plataforma .NET. La versión 2.1 introdujo la biblioteca HTTPClientFactory para crear instancias de cliente HTTP para interactuar con recursos basados en direcciones URL. Reemplazando la clase HTTPClient original, la clase de fábrica admite muchas características mejoradas, una de las cuales es la integración estrecha con la biblioteca de resiliencia Polly. Con él, puede definir fácilmente directivas de resistencia en la clase de inicio de la aplicación para controlar errores parciales y problemas de conectividad.

A continuación, vamos a expandir los patrones de reintentos y disyuntores.

Patrón Retry

En un entorno nativo de nube distribuido, las llamadas a servicios y recursos en la nube pueden producir errores debido a errores transitorios (de corta duración), que normalmente se corrigen después de un breve período de tiempo. La implementación de una estrategia de reintento ayuda a un servicio nativo en la nube a mitigar estos escenarios.

El patrón Retry permite a un servicio reintentar una operación de solicitud con error un número de veces (configurable) con un tiempo de espera exponencialmente creciente. En la figura 6-2 se muestra un reintento en acción.

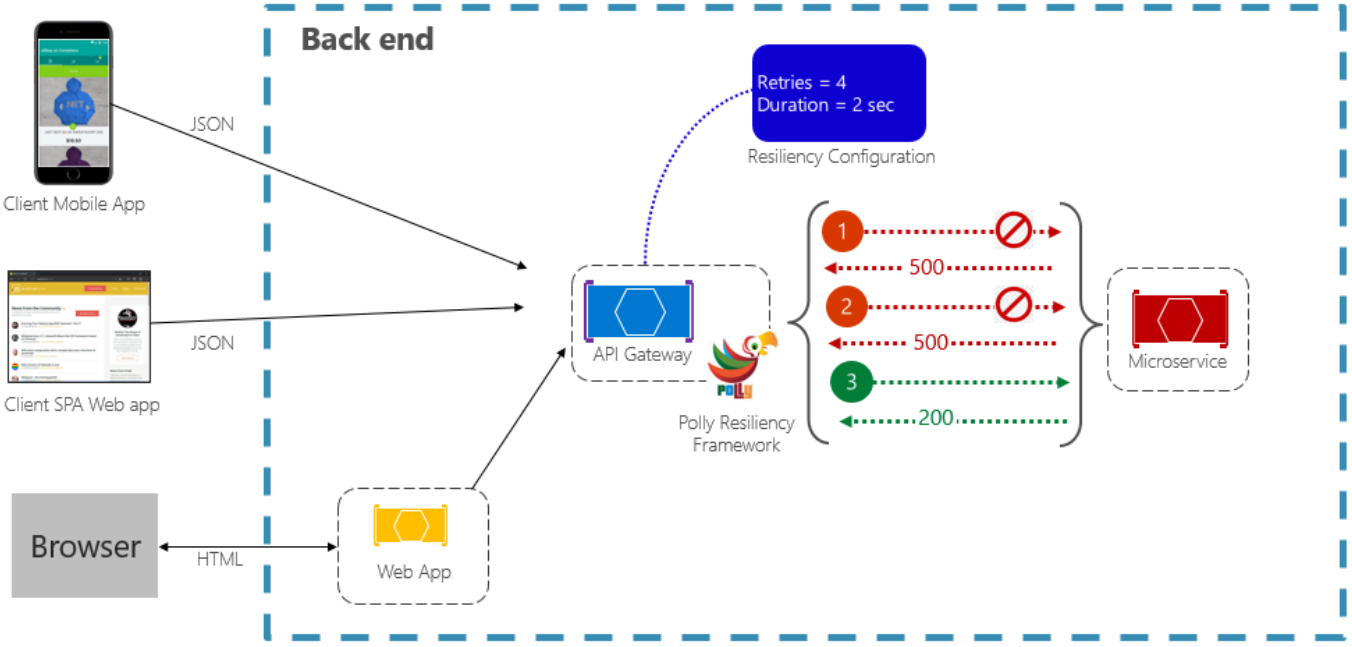


Figura 6-2. Patrón de reintento en acción

En la ilustración anterior, se ha implementado un patrón de reintento para una operación de solicitud. Está configurado para permitir hasta cuatro reintentos antes de que se produzca un error con un intervalo de retroceso (tiempo de espera) a partir de dos segundos, que se duplica exponencialmente para cada intento posterior.

- Se produce un error en la primera invocación y se devuelve un código de estado HTTP de 500. La aplicación espera dos segundos y vuelve a intentar la llamada.
- La segunda invocación también produce un error y devuelve un código de estado HTTP de 500. La aplicación ahora duplica el intervalo de retroceso a cuatro segundos y vuelve a intentar la llamada.
- Por último, la tercera llamada tiene éxito.
- En este escenario, la operación de reintento habría intentado hasta cuatro reintentos al duplicar la duración del retroceso antes de producir un error en la llamada.
- Si se produjo un error en el 4º intento de reintento, se invocaría una directiva de reserva para controlar correctamente el problema.

Es importante aumentar el período de retroceso antes de reintentar la llamada para permitir que el tiempo de servicio sea autocorrección. Es un procedimiento recomendado implementar

un retroceso exponencialmente creciente (duplicando el período en cada reintento) para permitir un tiempo de corrección adecuado.

Patrón de disyuntor

Aunque el patrón de reintento puede ayudar a salvar una solicitud entrelazada en un error parcial, hay situaciones en las que los errores pueden deberse a eventos imprevistos que requerirán períodos de tiempo más largos para resolverse. La gravedad de estos errores puede abarcar desde una pérdida parcial de la conectividad hasta la total detención de un servicio. En estas situaciones, es inútil que una aplicación vuelva a intentar continuamente una operación que es poco probable que se realice correctamente.

Para empeorar las cosas, la ejecución de operaciones continuas de reintento en un servicio no con capacidad de respuesta puede moverle a un escenario de denegación de servicio autoimpuesto en el que inunda el servicio con llamadas continuas que agotan recursos, como memoria, subprocesos y conexiones de base de datos, lo que provoca errores en partes no relacionadas del sistema que usan los mismos recursos.

En estas situaciones, sería preferible que la operación falle inmediatamente y solo intente invocar el servicio si es probable que tenga éxito.

El patrón Circuit Breaker puede impedir que una aplicación intente ejecutar repetidamente una operación que probablemente produzca un error. Después de un número predefinido de llamadas erróneas, bloquea todo el tráfico al servicio. Periódicamente, se realizará una llamada de prueba para determinar si la falla se ha resuelto. En la figura 6-3 se muestra el patrón Circuit Breaker en acción.

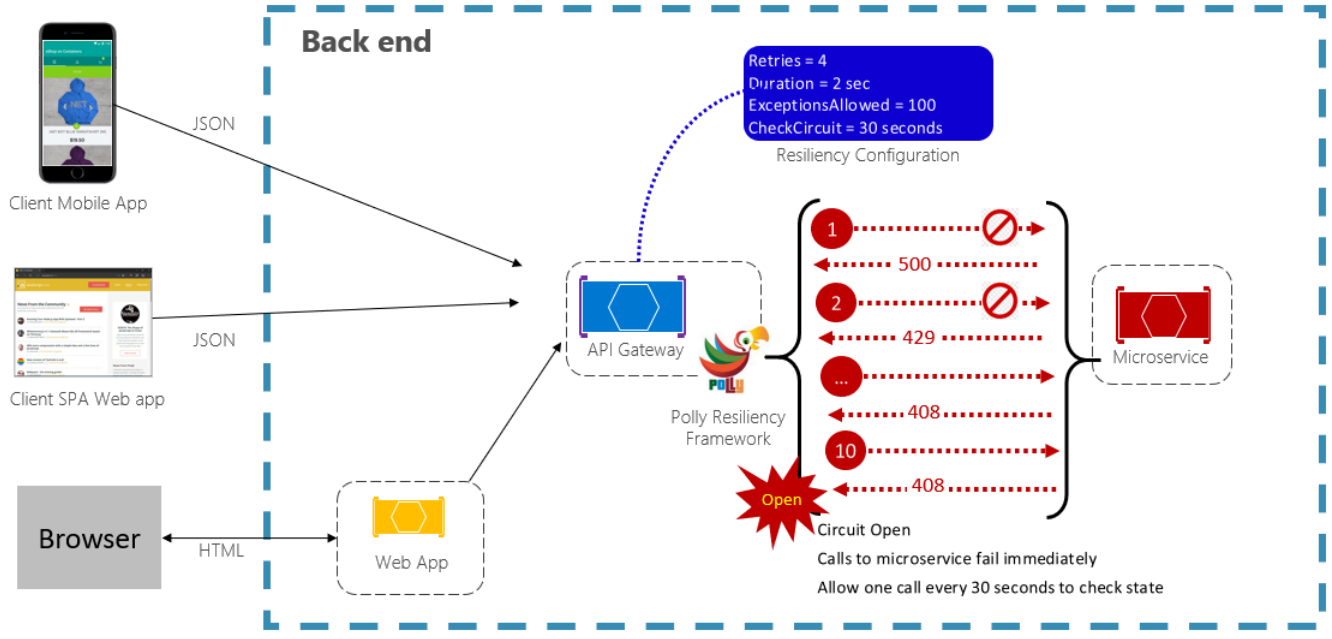


Figura 6-3. Patrón de disyuntor en acción

En la ilustración anterior, se ha agregado un patrón de disyuntor al patrón de reintento original. Observe cómo después de 100 solicitudes con error, los disyuntores se abren y ya no permiten llamadas al servicio. El valor CheckCircuit, establecido en 30 segundos, especifica la frecuencia con la que la biblioteca permite que una solicitud continúe con el servicio. Si esa llamada se realiza correctamente, el circuito se cierra y el servicio vuelve a estar disponible para el tráfico.

Tenga en cuenta que la intención del patrón Circuit Breaker es *diferente* de la del patrón Retry. El patrón Retry permite a una aplicación reintentar una operación con la expectativa de que se realizará correctamente. El patrón Circuit Breaker impide que una aplicación realice una operación que probablemente produzca un error. Normalmente, una aplicación *combinará* estos dos patrones mediante el patrón Retry para invocar una operación a través de un disyuntor.

Pruebas de resistencia

Las pruebas de resistencia no siempre se pueden realizar de la misma manera que se prueba la funcionalidad de la aplicación (mediante la ejecución de pruebas unitarias, pruebas de integración, etc.). En su lugar, debe probar cómo se comporta la carga de trabajo de extremo a extremo en condiciones de fallo, que solo se producen de forma intermitente. Por ejemplo: insertar errores bloqueando procesos, certificados expirados, hacer que los servicios dependientes no estén disponibles, etc. Los marcos como chaos-monkey se pueden usar para estas pruebas de caos.

La resistencia de la aplicación es un elemento necesario para controlar las operaciones solicitadas problemáticas. Pero es sólo la mitad de la historia. A continuación, tratamos las características de resistencia disponibles en la nube de Azure.

Siguiente anterior

Resistencia de la plataforma Azure

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico “Architecting Cloud Native .NET Applications for Azure” (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

La compilación de una aplicación confiable en la nube es diferente al desarrollo clásico de las aplicaciones locales. Aunque históricamente se compraba hardware de gama superior para escalar verticalmente, en un entorno de nube se escala horizontalmente. En vez de intentar evitar los errores, el objetivo es minimizar sus efectos y que el sistema se mantenga estable.

Dicho esto, las aplicaciones confiables en la nube tienen varias características distintivas:

- Son resistentes, se recuperan limpiamente de los problemas y siguen funcionando.
- Tienen alta disponibilidad y se ejecutan según lo previsto en un estado correcto, sin tiempos de inactividad significativos.

Comprender cómo funcionan juntas estas características (y cómo influyen en el coste) es esencial para compilar una aplicación confiable y nativa de nube. A continuación, veremos las formas en que puede compilar resistencia y disponibilidad en las aplicaciones nativas de nube, gracias al provecho que puede sacar de las características de la nube de Azure.

Diseño con resistencia

Hemos dicho que la resistencia permite que la aplicación reaccione ante errores y siga siendo funcional. En las Notas del producto para la resistencia en Azure se dan instrucciones para lograr resistencia en la plataforma Azure. Estas son algunas recomendaciones fundamentales:

- *Errores de hardware*. Compile redundancia en la aplicación mediante la implementación de componentes en distintos dominios de error. Por ejemplo, asegúrese de que las máquinas virtuales de Azure se colocan en bastidores diferentes mediante Conjuntos de disponibilidad.

- *Errores de los centros de datos*. Compile redundancia en la aplicación con zonas de aislamiento de errores en los centros de datos. Por ejemplo, asegúrese de que las máquinas virtuales de Azure se colocan en distintos centros de datos con aislamiento por error mediante Azure Availability Zones.

- *Errores en regiones*. Replique los datos y los componentes en otra región para que las aplicaciones se puedan recuperar rápidamente. Por ejemplo, use Azure Site Recovery para replicar máquinas virtuales de Azure en otra región de Azure.

- *Carga pesada*. Equilibre la carga en las instancias para controlar los picos de uso. Por ejemplo, coloque dos o más máquinas virtuales de Azure detrás de un equilibrador de carga para distribuir el tráfico por todas las máquinas virtuales.

- *Eliminación accidental o daños en los datos*. Realice una copia de seguridad de los datos para que se puedan restaurar si se producen eliminaciones o daños. Por ejemplo, use Azure Backup para realizar copias de seguridad periódicas de las máquinas virtuales de Azure.

Diseño con redundancia

Los errores varían en el ámbito del impacto. Un error de hardware, como un disco con errores, puede afectar a un nodo único de un clúster. Un error de conmutador de red podría afectar a toda una estantería de servidor. Los errores menos habituales, como los cortes de luz, podrían interrumpir la actividad de un centro de datos completo. Es aún más improbable que una región deje de estar disponible.

La redundancia es una forma de proporcionar resistencia a la aplicación. El nivel exacto de redundancia necesaria depende de los requisitos empresariales y afectará tanto al coste como a la complejidad del sistema. Por ejemplo, las implementaciones en varias regiones son más caras y más complicadas de administrar que las de una sola región. Necesitará procedimientos operativos para administrar la conmutación por error y la

conmutación por recuperación. El coste adicional y la complejidad podrían estar justificados para algunos escenarios empresariales, pero no para otros.

Para diseñar la arquitectura de la redundancia... ¿Debe identificar las rutas críticas de la aplicación y, después, determinar si hay redundancia en cada punto de la ruta de acceso? Si se produce un error en un subsistema, ¿la aplicación conmutará por error a otro elemento? Por último, necesita una comprensión clara de esas características integradas en la plataforma en la nube de Azure, para poder sacar provecho de ellas y así cumplir con los requisitos de redundancia. Estas son algunas recomendaciones para diseñar la arquitectura de la redundancia:

- *Implemente varias instancias de servicios.* Si la aplicación depende de una única instancia de un servicio, crea un único punto de error. El aprovisionamiento de varias instancias mejora tanto la resistencia como la escalabilidad. Al hospedar en Azure Kubernetes Service, puede configurar mediante declaración instancias redundantes (conjuntos de réplicas) en el archivo de manifiesto de Kubernetes. El valor del recuento de réplicas se puede administrar mediante programación, en el portal o mediante características de escalado automático.

- *Sacar provecho de un equilibrador de carga.* El equilibrio de carga distribuye las solicitudes de la aplicación en instancias de servicio correctas y elimina automáticamente de la rotación las instancias incorrectas. Al hacer la implementación en Kubernetes, el equilibrio de carga se puede especificar en el archivo de manifiesto de Kubernetes, en la sección Servicios.

- *Planificación de la implementación en varias regiones.* Si implementa su aplicación en una sola región y dicha región deja de estar disponible, la aplicación tampoco estará disponible. Esto puede ser inaceptable según los términos de los contratos de nivel de servicio de su aplicación. Para evitar esa situación, considere la posibilidad de implementar la aplicación y sus servicios en varias regiones. Por ejemplo, un clúster de Azure Kubernetes Service (AKS) se implementa en una sola región. Para proteger el sistema de un error que se deba a la región, puede implementar la aplicación en varios clústeres de AKS, en distintas regiones, y usar la característica Regiones emparejadas para coordinar las actualizaciones de la plataforma y priorizar los esfuerzos de recuperación.

- *Habilitación de la replicación geográfica.* La replicación geográfica para servicios como Azure SQL Database y Cosmos DB creará réplicas secundarias de los datos en varias regiones. Aunque ambos servicios replicarán automáticamente los datos dentro de la misma región, la replicación geográfica le protege contra una interrupción regional, ya que le permite conmutar por error a una región secundaria. Otro procedimiento recomendado para la replicación geográfica se

centra en el almacenamiento de imágenes de contenedor. Para implementar un servicio en AKS, debe almacenar y extraer la imagen de un repositorio. Azure Container Registry se integra con AKS y puede almacenar imágenes de contenedor de forma segura. Para mejorar el rendimiento y la disponibilidad, considere la opción de replicar geográficamente sus imágenes en un registro de cada región en la que haya un clúster de AKS. Así, cada clúster de AKS extrae imágenes de contenedor del registro de contenedor local que se halle en su región, como se muestra en la figura 6-4:

Figura 6-4. Recursos replicados en varias regiones

- *Implemente un equilibrador de carga de tráfico DNS.*Azure Traffic Manager proporciona alta disponibilidad para las aplicaciones críticas mediante el equilibrio de carga en el nivel DNS. Puede enrutar el tráfico a diferentes regiones en función de la geografía, el tiempo de respuesta del clúster e incluso el estado del punto de conexión de la aplicación. Por ejemplo, Azure Traffic Manager puede dirigir a los clientes a la instancia de la aplicación y al clúster de AKS más próximos. Si tiene varios clústeres de AKS en distintas regiones, use Traffic Manager para controlar cómo fluye el tráfico a las aplicaciones que se ejecutan en cada clúster. En la figura 6-5 se muestra este escenario.

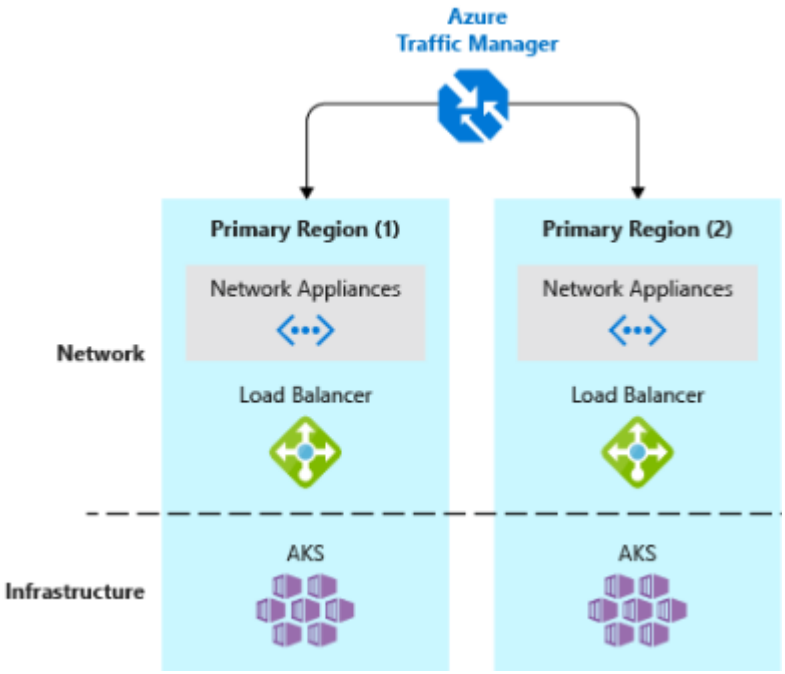


Figura 6-5. AKS y Azure Traffic Manager

Diseño para escalabilidad

La nube se basa en el escalado. La capacidad de aumentar o disminuir los recursos del sistema para abordar la carga creciente o decreciente del sistema es un principio clave de la nube de Azure. Sin embargo, para escalar eficazmente una aplicación, necesita comprender las características de escalado de cada servicio de Azure que incluya en la aplicación. Estas son recomendaciones para implementar de forma eficaz el escalado en el sistema.

- *Diseño para el escalado.* Una aplicación debe diseñarse para el escalado. Para iniciarse, los servicios deben estar sin estado para que las solicitudes se puedan enrutar a cualquier instancia. Tener servicios sin estado también significa que la adición o eliminación de una instancia no supone efectos negativos para los usuarios actuales.

- *Cargas de trabajo de particiones*. La descomposición de dominios en microservicios independientes y autosuficientes permite que cada servicio se escale independientemente de otros. Normalmente, los servicios tendrán necesidades y requisitos de escalabilidad diferentes. La creación de particiones permite escalar solo lo que se debe escalar, sin el coste innecesario de escalar una aplicación completa.

- *Se prioriza el escalado horizontal.* Las aplicaciones basadas en la nube priorizan el escalado horizontal de los recursos, en detrimento del escalado vertical. El escalado horizontal implica agregar más recursos de servicio a un sistema

existente para satisfacer y compartir un nivel de rendimiento deseado. El escalado vertical implica reemplazar los recursos existentes por un hardware más eficaz (más discos, más memoria y más núcleos de procesamiento). El escalado horizontal se puede invocar automáticamente con las características de escalado automático disponibles en algunos recursos de la nube de Azure. El escalado horizontal en varios recursos también agrega redundancia al sistema en su totalidad. Por último, el escalado vertical de un único recurso suele ser más caro que el escalado horizontal en muchos recursos más pequeños. En la figura 6-6 se muestran los dos enfoques:

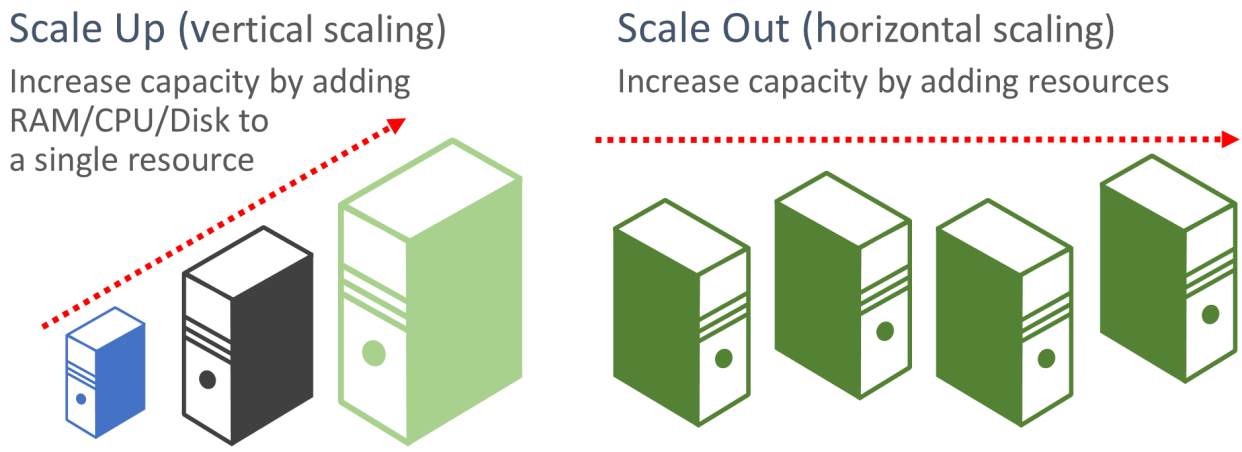


Figura 6-6. Escalado vertical versus escalado horizontal

- *Escale proporcionalmente.* Al escalar un servicio, piense en base a *conjuntos de recursos*. Si escala considerablemente un servicio específico, ¿qué impacto tendrá en los almacenes de datos de back-end, las memorias caché y los servicios dependientes? Algunos recursos, como Cosmos DB, pueden escalar horizontalmente de manera proporcional, pero muchos otros no. Es bueno asegurarse de que un recurso no se escale horizontalmente hasta un punto en el que agote otros recursos asociados.

- *Evite la afinidad.* Un procedimiento recomendado es asegurarse de que un nodo no requiere afinidad local, a menudo denominada "*sticky session*" (sesión fija). Una solicitud debe ser capaz de enrutarse a cualquier instancia. Si necesita conservar el estado, este debe guardarse en una caché distribuida, como Azure Redis Cache.

- *Aprovechar las características de escalado automático de la plataforma.* Use las características de escalado automático integradas siempre que sea posible, en vez de mecanismos personalizados o de terceros. Siempre que sea posible, use reglas de escalado programadas para garantizar que los recursos están disponibles sin retrasos iniciales, pero agregue escalado automático reactivo a las reglas, cuando sea apropiado, para hacer frente a cambios inesperados en la demanda. Para más información, consulte Guía de escalado automático.

- *Escalado horizontal repentino.* Una última práctica es escalar horizontalmente de forma repentina para poder satisfacer rápidamente picos inmediatos en el tráfico sin perder actividad del negocio. Después, se reduce horizontalmente (es decir, se quitan instancias innecesarias) con cautela para mantener el sistema estable. Una manera sencilla de implementar esto es usando el período de recuperación, que es el tiempo de espera entre las operaciones de escalado. Establézcalo en cinco minutos para agregar recursos y en 15 minutos como tope para quitar instancias.

Reintento integrado en los servicios

Uno de los procedimientos recomendados que mencionamos en una sección anterior es la implementación de operaciones de reintento en la programación. Tenga en cuenta que muchos servicios de Azure y sus SDK de cliente correspondientes también incluyen mecanismos de reintento. En la lista siguiente se resumen las características de reintento de los muchos servicios de Azure que se describen en este libro:

- *Azure Cosmos DB.* La clase DocumentClient de la API de cliente retira automáticamente los intentos erróneos. El número de reintentos y el tiempo de espera máximo son configurables. Las excepciones producidas por la API de cliente son solicitudes que superan lo marcado en la directiva de reintentos o errores no transitorios.

- *Azure Redis Cache.* El cliente de Redis StackExchange usa una clase de administrador de conexiones que incluye reintentos de los intentos erróneos. El número de reintentos, la directiva de reintentos específica y el tiempo de espera son configurables.

- *Azure Service Bus.* El cliente de Service Bus expone una clase RetryPolicy que se puede configurar con un intervalo de espera, un recuento de reintentos y TerminationTimeBuffer, que especifica el tiempo máximo que puede tardar una operación. La directiva predeterminada es nueve reintentos como máximo, con un período de espera de 30 segundos entre intentos.

- *Azure SQL Database.* Se proporciona compatibilidad con reintentos al usar la biblioteca de Entity Framework Core.

- *Azure Storage.* La biblioteca cliente de Storage admite operaciones de reintento. Las estrategias varían entre tablas, blobs y colas de Azure Storage. Además, los reintentos alternativos cambian entre las ubicaciones de los servicios de almacenamiento principal y secundario cuando se habilita la característica de redundancia geográfica.

- *Azure Event Hubs*. La biblioteca cliente del Centro de eventos cuenta con una propiedad RetryPolicy, que incluye una característica de retroceso exponencial configurable.

Anterior Siguiente

Comunicaciones resistentes

Artículo • 08/05/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

A lo largo de este libro, hemos adoptado un enfoque arquitectónico basado en microservicios. Aunque esta arquitectura proporciona ventajas importantes, presenta muchos desafíos:

- *Comunicación de red fuera de proceso.* Cada microservicio se comunica a través de un protocolo de red que introduce congestión de red, latencia y errores transitorios.
- *Detección de servicios.* ¿Cómo detectan y comunican los microservicios entre sí al ejecutarse en un clúster de máquinas con sus propias direcciones IP y puertos?
- *Resistencia.* ¿Cómo administra los errores de corta duración y mantiene el sistema estable?
- *Equilibrio de carga.* ¿Cómo se distribuye el tráfico entrante entre varias instancias de un microservicio?
- *Seguridad.* ¿Cómo se aplican los problemas de seguridad, como el cifrado de nivel de transporte y la administración de certificados?
- *Supervisión distribuida.* - ¿Cómo correlaciona y captura la rastreabilidad y la supervisión de una sola solicitud en varios microservicios de consumo?

Puede abordar estos problemas con diferentes bibliotecas y marcos, pero la implementación puede ser costosa, compleja y lenta. También termina con problemas de infraestructura acoplados a la lógica de negocios.

Malla de servicio

Un mejor enfoque es una tecnología en evolución titulada *Service Mesh*. Una malla de servicio es una capa de infraestructura configurable con funcionalidades integradas para controlar la comunicación del servicio y los otros desafíos mencionados anteriormente. Desacopla estos problemas al moverlos a un proxy de servicio. El proxy se implementa en un proceso independiente (denominado sidecar) para proporcionar aislamiento del código empresarial. Sin embargo, el sidecar está vinculado al servicio: se crea con él y comparte su ciclo de vida. En la figura 6-7 se muestra este escenario.

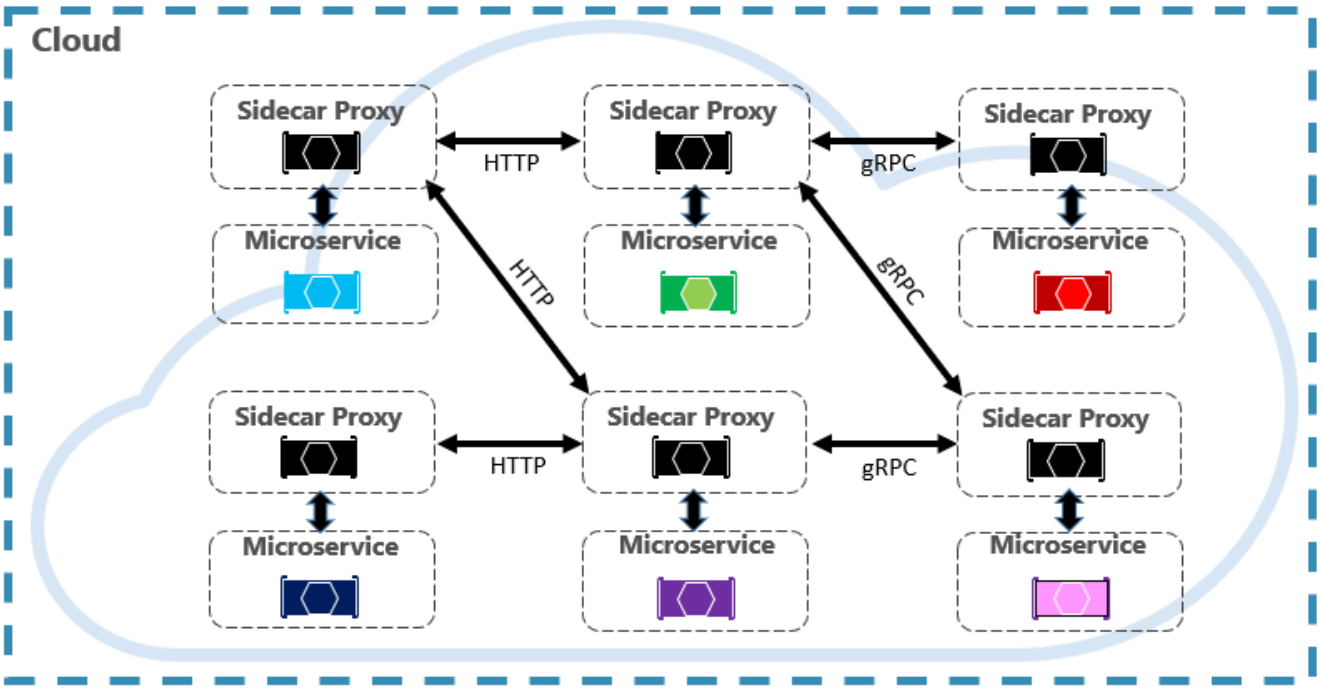


Figura 6-7. Malla de servicio con un coche lateral

En la ilustración anterior, observe cómo el proxy intercepta y administra la comunicación entre los microservicios y el clúster.

Una malla de servicio se divide lógicamente en dos componentes dispares: un plano de datos y un plano de control. En la figura 6-8 se muestran estos componentes y sus responsabilidades.

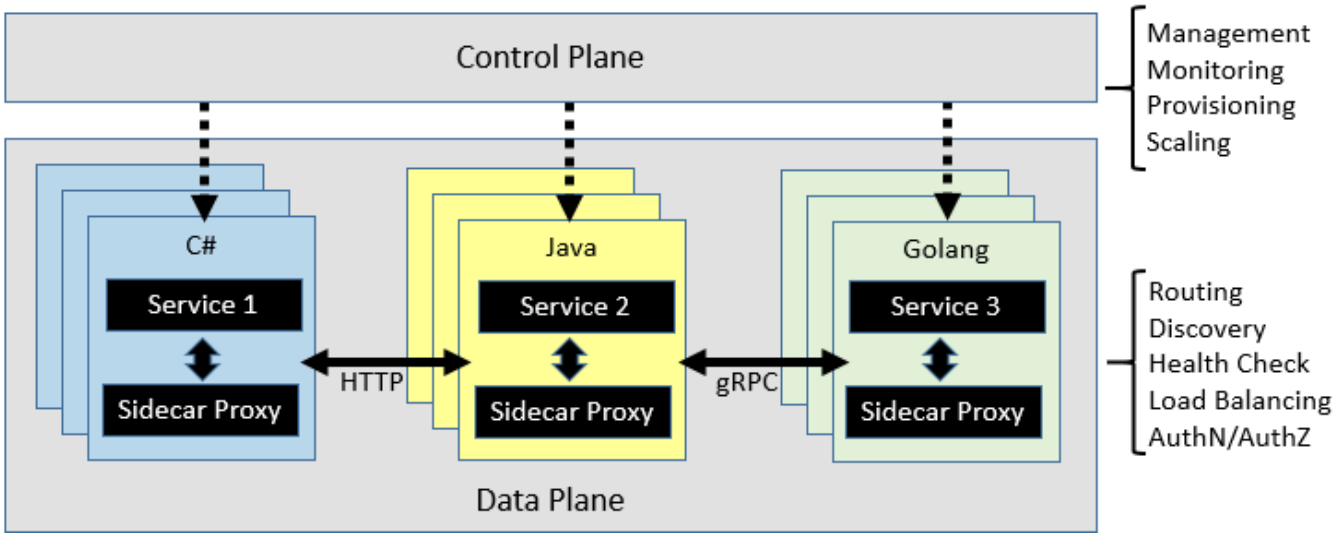


Figura 6-8. Control de malla de servicio y plano de datos

Una vez configurada, una malla de servicio es muy funcional. Puede recuperar un grupo de instancias correspondiente de un punto de conexión de detección de servicios. Después, la malla puede enviar una solicitud a una instancia específica, registrando la latencia y el tipo de respuesta del resultado. Una malla puede elegir la instancia con más probabilidades de devolver una respuesta rápida en función de muchos factores, incluida su latencia observada para las solicitudes recientes.

Si una instancia no responde o no responde, la malla reintentará la solicitud en otra instancia. Si devuelve errores, una malla expulsará la instancia del grupo de equilibrio de carga y la restablezará después de que se sane. Si se agota el tiempo de espera de una solicitud, se puede producir un error en una malla y, a continuación, volver a intentar la solicitud. Una malla captura y emite métricas y seguimiento distribuido a un sistema de métricas centralizado.

Istio y Envoy

Aunque actualmente existen algunas opciones de malla de servicio, Istio es la más popular en el momento de escribir este documento. Istio es una empresa conjunta de IBM, Google y Lyft. Es una oferta de código abierto que se puede integrar en una aplicación distribuida nueva o existente. La tecnología proporciona una solución coherente y completa para proteger, conectar y supervisar microservicios. Sus características incluyen:

- Protección de la comunicación entre servicios en un clúster con autenticación y autorización sólidas basadas en identidades.
- Equilibrio de carga automático para el tráfico HTTP, gRPC, WebSocket y TCP.
- Control específico del comportamiento del tráfico, con reglas de enrutamiento enriquecidas, reintentos, conmutaciones por error e inyección de errores
- Una API de configuración y capa de directiva conectable que admite controles de acceso, límites de velocidad y cuotas

- Métricas, registros y seguimientos automáticos para todo el tráfico de un clúster, incluida la entrada y salida del clúster

Un componente clave para una implementación de Istio es un servicio proxy titulado proxy proxy. Se ejecuta junto con cada servicio y proporciona una base independiente de la plataforma para las siguientes características:

- Detección dinámica de servicios.
- Equilibrio de carga.
- Terminación TLS.
- Servidores proxy HTTP y gRPC.
- Resistencia del disyuntor.
- Comprobaciones de estado.
- Actualizaciones graduales con implementaciones controladas.

Como se explicó anteriormente, Envoy se implementa como sidecar en cada microservicio del clúster.

Integración con Azure Kubernetes Services

La nube de Azure adopta Istio y proporciona compatibilidad directa con ella en Azure Kubernetes Services. Los vínculos siguientes pueden ayudarle a empezar:

- Instalación de Istio en AKS
- Uso de AKS e Istio

Referencias

- Polly
- Patrón de reintento
- Patrón Circuit Breaker
- Notas del producto sobre resistencia en Azure
- Redundancia
- replicación geográfica
- Azure Traffic Manager
- Guía de escalado automático
- Istio

- Proxy de Envoy

Siguiente anterior

Supervisión y mantenimiento

17/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Los microservicios y las aplicaciones nativas de la nube van de la mano con buenas prácticas de DevOps. DevOps es muchas cosas para muchas personas, pero quizás una de las mejores definiciones proviene del promotor de la nube y del pastor de DevOps Donovan Brown:

"DevOps es la unión de personas, procesos y productos para permitir la entrega continua de valor a nuestros usuarios finales".

Desafortunadamente, con definiciones breves, siempre hay espacio para decir más cosas. Uno de los componentes clave de DevOps es garantizar que las aplicaciones que se ejecutan en producción funcionen correctamente y de forma eficaz. Para medir el estado de la aplicación en producción, es necesario supervisar los distintos logs y métricas que se generan a partir de los servidores, hosts y la propia aplicación. El número de servicios diferentes que se ejecutan en compatibilidad con una aplicación nativa de la nube hace que la supervisión del estado de los componentes individuales y la aplicación en su conjunto sea un desafío crítico.

Siguiente anterior

Patrones de observabilidad

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Al igual que se han desarrollado patrones para ayudar en el diseño de código en las aplicaciones, hay patrones para las aplicaciones operativas de una manera confiable. Han surgido tres patrones útiles para mantener aplicaciones: **registro**, **supervisión** y **alertas**.

Cuándo usar el registro

Independientemente de lo cuidadoso que seamos, las aplicaciones casi siempre se comportan de maneras inesperadas en producción. Cuando los usuarios notifican problemas con una aplicación, resulta útil poder ver lo que estaba ocurriendo con la aplicación cuando se produjo el problema. Una de las formas más probadas y verdaderas de capturar información sobre lo que hace una aplicación mientras se ejecuta es hacer que la aplicación anote lo que está haciendo. Este proceso se conoce como registro. Siempre que se produzcan errores o problemas en producción, el objetivo debe ser reproducir las condiciones en las que se produjeron los errores, en un entorno que no sea de producción. Contar con un buen sistema de registro proporciona una hoja de ruta para que los desarrolladores puedan duplicar problemas en un entorno que permite realizar pruebas y experimentación.

Desafíos al iniciar sesión con aplicaciones nativas de la nube

En las aplicaciones tradicionales, los archivos de registro se almacenan normalmente en el equipo local. De hecho, en sistemas operativos similares a Unix, hay una estructura de carpetas

definida para contener los registros, normalmente en `/var/log`.

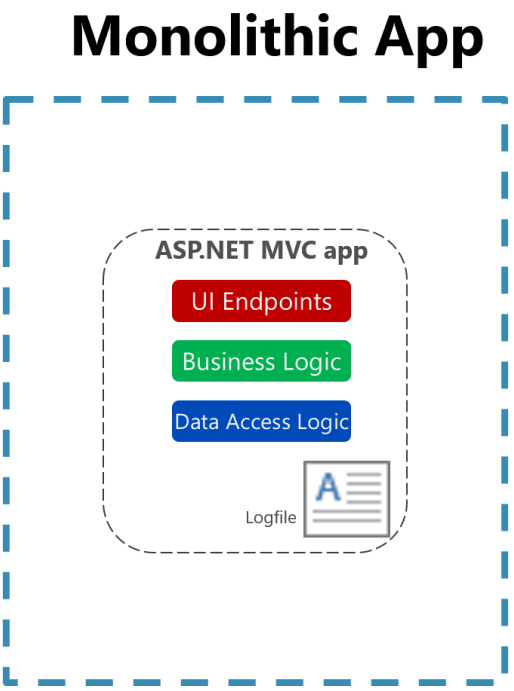


Figura 7-1. Registro en un archivo en una aplicación monolítica.

La utilidad de registrar en un archivo plano en una sola máquina se reduce enormemente en un entorno de nube. Es posible que las aplicaciones que producen registros no tengan acceso al disco local o que el disco local sea muy transitorio, ya que los contenedores se ordenan aleatoriamente en torno a las máquinas físicas. Incluso un simple escalado vertical de aplicaciones monolíticas en varios nodos puede dificultar la localización de un archivo de registro basado en archivos.

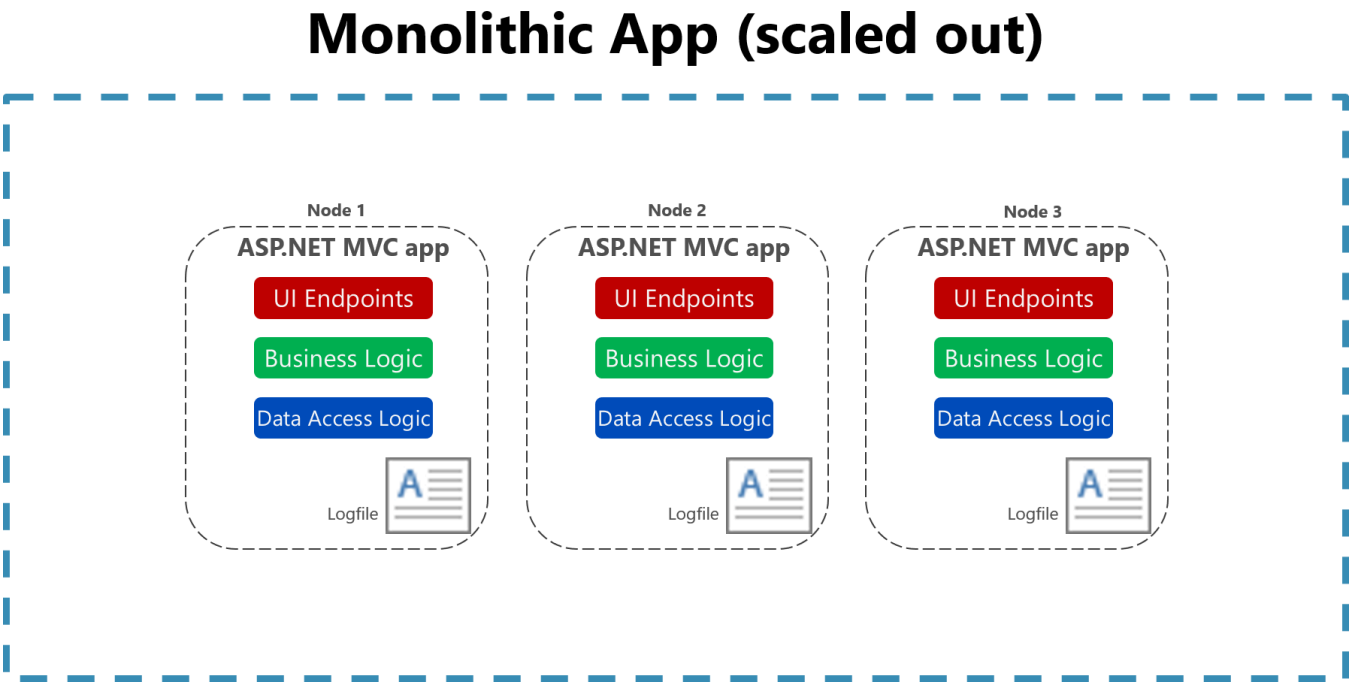


Figura 7-2. Registro en archivos en una aplicación monolítica escalada.

Las aplicaciones nativas de la nube desarrolladas mediante una arquitectura de microservicios también suponen algunos desafíos para los registradores basados en archivos. Las solicitudes de usuario ahora pueden abarcar varios servicios que se ejecutan en máquinas diferentes y pueden incluir funciones sin servidor sin acceso a un sistema de archivos local en absoluto. Sería muy difícil correlacionar los registros de un usuario o una sesión en estos muchos servicios y máquinas.

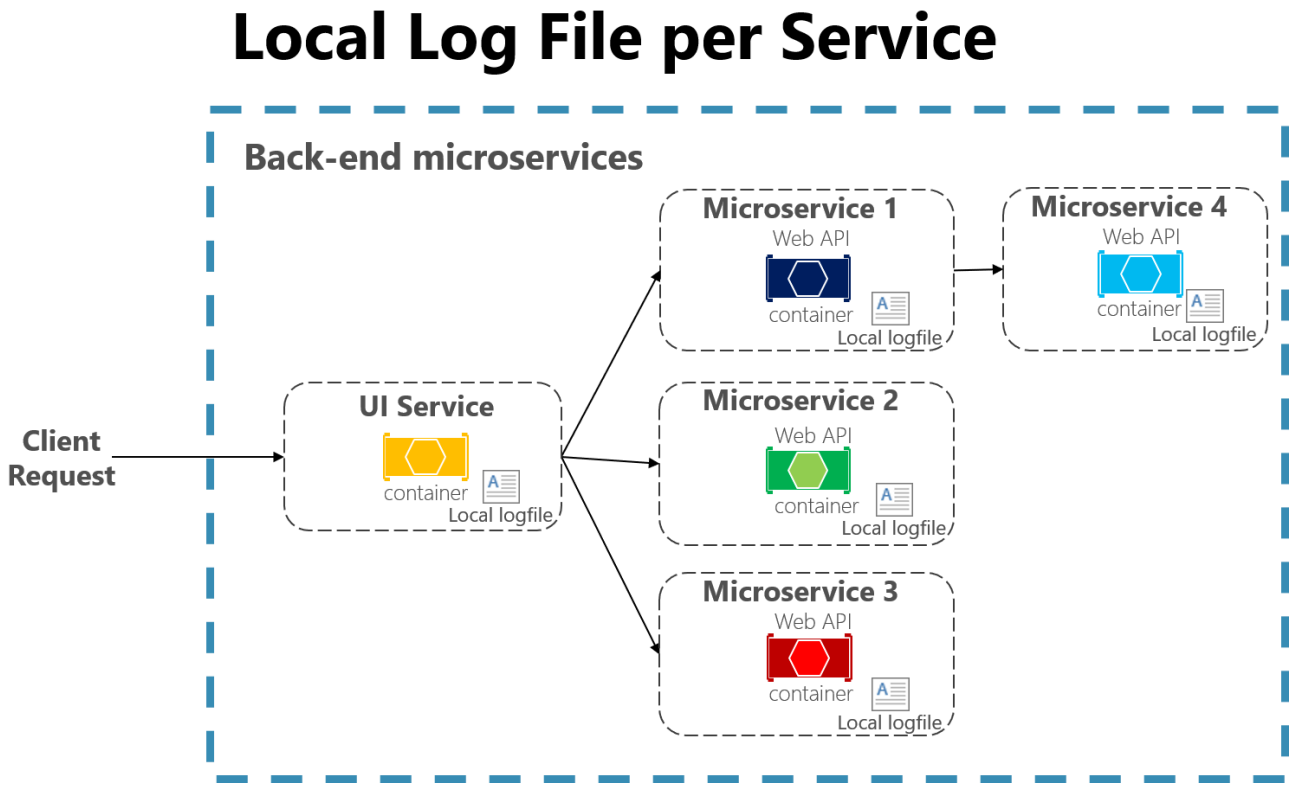


Figura 7-3. Registro en archivos locales en una aplicación de microservicios.

Por último, el número de usuarios de algunas aplicaciones nativas de la nube es elevado. Imagine que cada usuario genera cien líneas de mensajes de registro cuando inicia sesión en una aplicación. De forma aislada, es manejable, pero multiplica eso por 100 000 usuarios y el volumen de registros se vuelve lo suficientemente grande como para que se necesiten herramientas especializadas para respaldar el uso eficaz de los registros.

Registro en aplicaciones nativas de la nube

Cada lenguaje de programación tiene herramientas que permiten escribir registros y, normalmente, la sobrecarga para escribir estos registros es baja. Muchas de las bibliotecas de registro permiten registrar diferentes niveles de criticidad, que se pueden ajustar en tiempo de ejecución. Por ejemplo, la biblioteca Serilog es una conocida biblioteca de registro estructurado para .NET que proporciona los siguientes niveles de registro:

- Verbose
- Depurar
- Información

- Advertencia
- Error
- Grave

Estos distintos niveles de registro proporcionan granularidad en el registro. Cuando la aplicación funciona correctamente en producción, puede configurarse para registrar solo mensajes importantes. Cuando la aplicación se comporta mal, se puede aumentar el nivel de registro para que se recopilen registros más detallados. Esto equilibra el rendimiento y la facilidad de depuración.

El alto rendimiento de las herramientas de registro y la capacidad de ajuste del nivel de detalle deben animar a los desarrolladores a registrar con frecuencia. Muchos favorecen un patrón de registro de la entrada y salida de cada método. Este enfoque puede parecer excesivo, pero es poco frecuente que los desarrolladores deseen menos registro. De hecho, no es raro realizar implementaciones con el único fin de agregar registros sobre un método problemático. Es mejor realizar demasiados registros que demasiados pocos. Algunas herramientas se pueden usar para proporcionar automáticamente este tipo de registro.

Debido a los desafíos asociados al uso de registros basados en archivos en aplicaciones nativas de nube, se prefieren los registros centralizados. Las aplicaciones recopilan los registros y se envían a una aplicación de registro central que indexa y almacena los registros. Esta clase de sistema puede ingerir decenas de gigabytes de registros todos los días.

También es útil seguir algunos procedimientos estándar al crear registros que abarquen muchos servicios. Por ejemplo, generar un identificador de correlación al principio de una interacción larga y, a continuación, registrarlo en cada mensaje relacionado con esa interacción facilita la búsqueda de todos los mensajes relacionados. Solo necesita encontrar un solo mensaje y extraer el identificador de correlación para buscar todos los mensajes relacionados. Otro ejemplo es asegurarse de que el formato de registro es el mismo para cada servicio, independientemente del idioma o la biblioteca de registro que use. Esta normalización facilita la lectura de registros. En la figura 7-4 se muestra cómo una arquitectura de microservicios puede aprovechar el registro centralizado como parte de su flujo de trabajo.

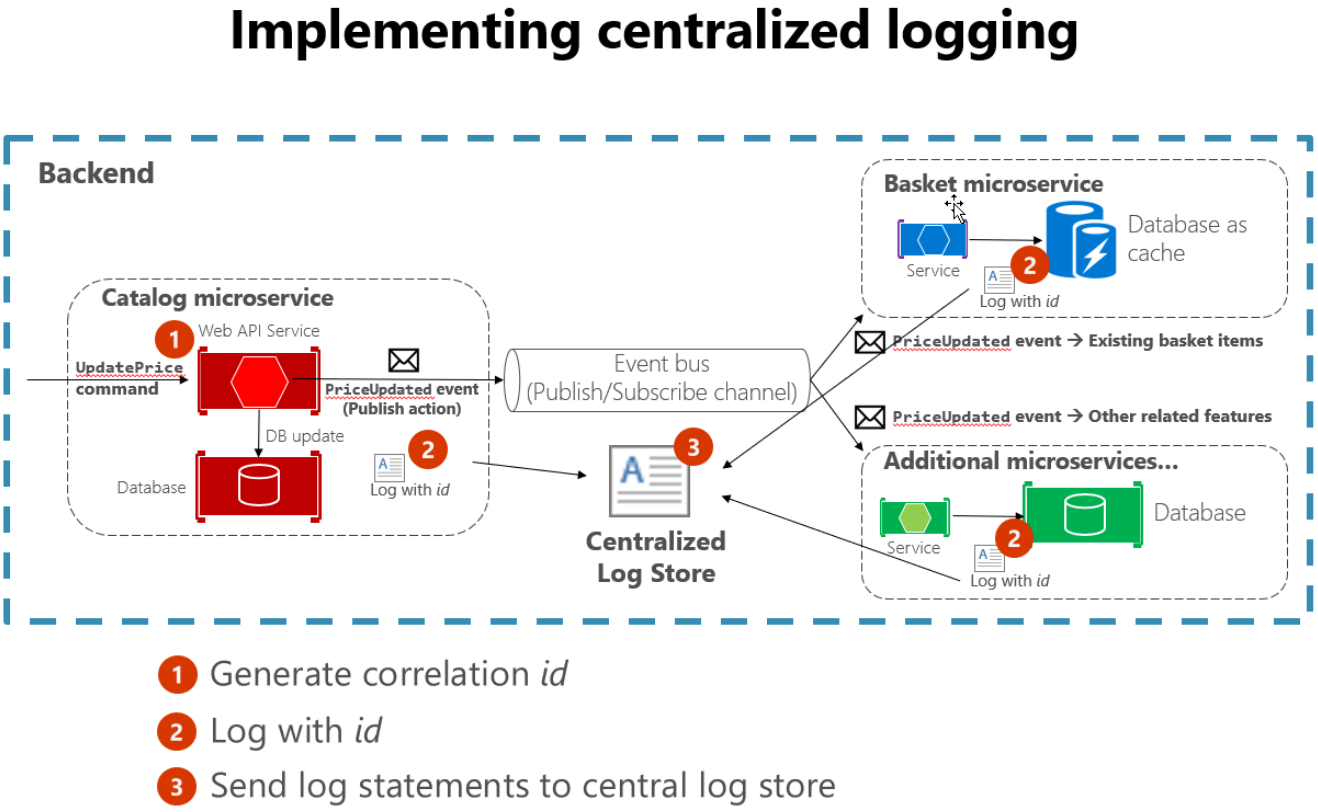


Figura 7-4. Los logs de varios orígenes se integran en un repositorio de logs centralizado.

Desafíos con la detección y respuesta a posibles problemas de estado de la aplicación

Algunas aplicaciones no son críticas. Quizá solo se usen internamente y, cuando se produzca un problema, el usuario puede ponerse en contacto con el equipo responsable y se puede reiniciar la aplicación. Sin embargo, los clientes suelen tener mayores expectativas para las aplicaciones que consumen. Debe saber cuándo se producen problemas con la aplicación *antes* de que los usuarios lo hagan o antes de que los usuarios le notifiquen. De lo contrario, la primera vez que se entere de un problema puede ser cuando se dé cuenta de un diluvio de publicaciones enfadadas en redes sociales que ridiculizan su aplicación o incluso su organización.

Algunos escenarios que puede tener en cuenta son:

- Un servicio de la aplicación sigue fallando y reiniciando, lo que da lugar a respuestas lentas intermitentes.
- En algunos momentos del día, el tiempo de respuesta de la aplicación es lento.
- Después de una implementación reciente, la carga en la base de datos se ha tripledo.

La supervisión se puede implementar correctamente para informarle sobre las condiciones que provocarán problemas, lo que le permite abordar las condiciones subyacentes antes de que produzcan cualquier impacto significativo en el usuario.

Supervisión de aplicaciones nativas de la nube

Algunos sistemas de registro centralizados tienen un rol adicional de recopilar telemetría fuera de los registros puros. Pueden recopilar métricas, como el tiempo para ejecutar una consulta de base de datos, el tiempo medio de respuesta de un servidor web e incluso los promedios de carga de CPU y la presión de memoria, según lo notificado por el sistema operativo. Junto con los registros, estos sistemas pueden proporcionar una vista holística del estado de los nodos en el sistema y la aplicación en su conjunto.

Las funcionalidades de recopilación de métricas de las herramientas de supervisión también se pueden alimentar manualmente desde dentro de la aplicación. Los flujos empresariales que son de especial interés, como los nuevos usuarios que se registran o los pedidos que se realizan, se pueden instrumentar de forma que incrementen un contador en el sistema de supervisión central. Este aspecto desbloquea las herramientas de supervisión para no solo supervisar el estado de la aplicación, sino el estado de la empresa.

Las consultas se pueden construir en las herramientas de agregación de registros para buscar determinadas estadísticas o patrones, que luego se pueden mostrar en formato gráfico, en paneles personalizados. Con frecuencia, los equipos invertirán en pantallas grandes y montadas en pared que giran a través de las estadísticas relacionadas con una aplicación. De este modo, es fácil ver los problemas a medida que se producen.

Las herramientas de supervisión nativas de la nube proporcionan telemetría en tiempo real e información sobre las aplicaciones, independientemente de si son aplicaciones monolíticas de un solo proceso o arquitecturas de microservicios distribuidos. Incluyen herramientas que permiten la recopilación de datos de la aplicación, así como herramientas para consultar y mostrar información sobre el estado de la aplicación.

Desafíos con la reacción a problemas críticos en aplicaciones nativas de la nube

Si necesita reaccionar ante problemas con la aplicación, necesita alguna manera de alertar al personal adecuado. Este es el tercer patrón de observabilidad de aplicaciones nativas de la nube y depende del registro y la supervisión. Tu aplicación necesita tener un registro de eventos para permitir el diagnóstico de problemas y, en algunos casos, alimentar herramientas de supervisión. Necesita supervisión para agregar métricas de aplicación y datos de estado en un solo lugar. Una vez establecido esto, se pueden crear reglas que desencadenarán alertas cuando determinadas métricas se encuentran fuera de los niveles aceptables.

Por lo general, las alertas se superponen a la supervisión, de modo que ciertas condiciones desencadenan alertas adecuadas para notificar a los miembros del equipo problemas urgentes.

Algunos escenarios que pueden requerir alertas incluyen:

- Uno de los servicios de la aplicación no responde después de 1 minuto de tiempo de inactividad.
- La aplicación devuelve respuestas HTTP incorrectas a más del 1 % de las solicitudes.
- El tiempo medio de respuesta de la aplicación para los puntos de conexión de clave supera los 2000 ms.

Alertas en aplicaciones nativas de la nube

Puede crear consultas en las herramientas de supervisión para buscar condiciones de error conocidas. Por ejemplo, las consultas podrían buscar en los registros entrantes indicaciones del código de estado HTTP 500, lo que indica un problema en un servidor web. Tan pronto como se detecte una de estas, se podría enviar un correo electrónico o un SMS al propietario del servicio de origen que puede empezar a investigar.

Sin embargo, normalmente, un único error 500 no es suficiente para determinar que se ha producido un problema. Podría significar que un usuario ha escrito mal su contraseña o ha escrito algunos datos con formato incorrecto. Las consultas de alerta pueden crearse de modo que solo se activen cuando se detecte un número superior a la media de errores 500.

Uno de los patrones más perjudiciales en las alertas es desencadenar demasiadas alertas para que los humanos investiguen. Los propietarios de servicios se dessensitizarán rápidamente a los errores que han investigado anteriormente y han detectado que son benignos. Después, cuando se produzcan errores de verdad, se perderán en el ruido que generan cientos de falsos positivos. La parábola de El niño que gritó 'lobo' es frecuentemente contada a los niños para advertirles de este peligro. Es importante asegurarse de que las alertas que se activan indiquen un problema real.

Siguiente anterior

Registro con pila elástica

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Hay muchas buenas herramientas de registro centralizadas y varían en el costo de las herramientas gratuitas y de código abierto, a opciones más costosas. En muchos casos, las herramientas gratuitas son tan buenas o mejores que las ofertas de pago. Una de estas herramientas es una combinación de tres componentes de código abierto: Elasticsearch, Logstash y Kibana.

En conjunto, estas herramientas se conocen como pila elástica o ELK.

Elastic Stack

Elastic Stack es una opción eficaz para recopilar información de un clúster de Kubernetes. Kubernetes admite el envío de registros a un punto de conexión de Elasticsearch y, por lo general, todo lo que necesita para empezar es establecer las variables de entorno como se muestra en la figura 7-5:

kubernetes

```
KUBE_LOGGING_DESTINATION=elasticsearch
KUBE_ENABLE_NODE_LOGGING=true
```

Figura 7-5. Variables de configuración para Kubernetes

Este paso instalará Elasticsearch en el clúster con el propósito de enviar todos los registros del clúster a Elasticsearch.

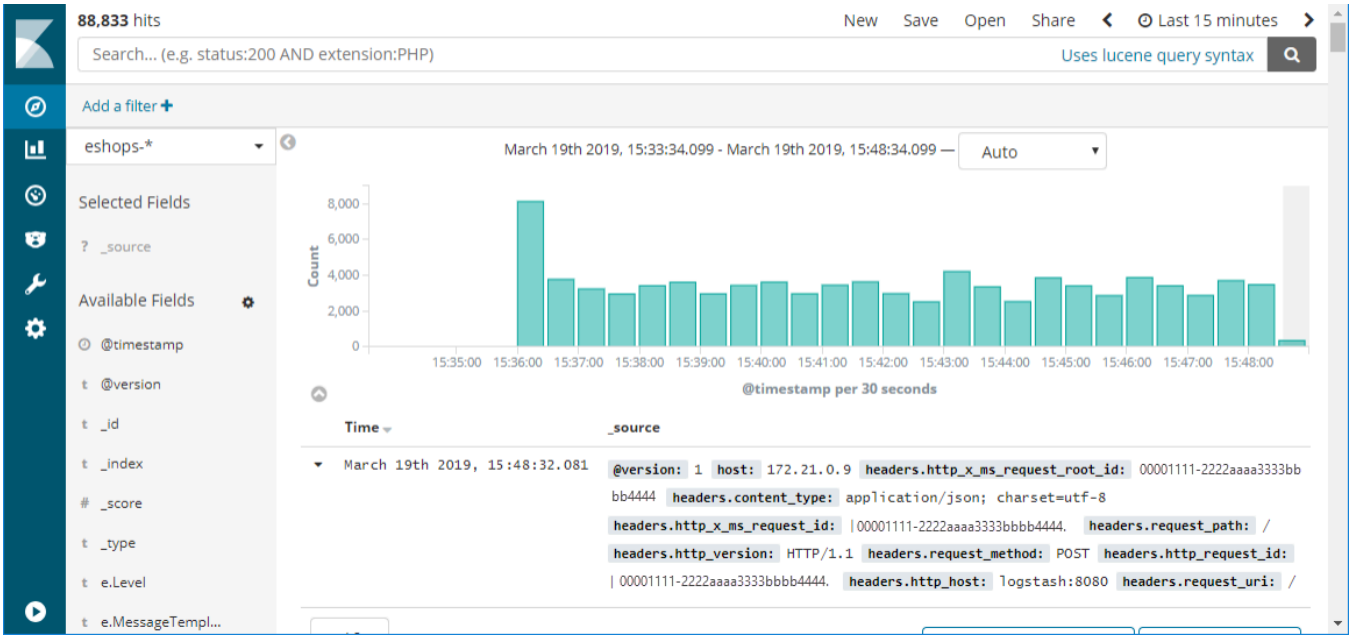


figura 7-6. Ejemplo de un panel de Kibana que muestra los resultados de una consulta en los registros que se ingieren desde Kubernetes

Para obtener más información sobre la configuración, consulte Configuración del registro (Kibana).

¿Cuáles son las ventajas de Elastic Stack?

Elastic Stack proporciona un registro centralizado de una manera fácil de usar en la nube, escalable y de bajo costo. Tu interfaz de usuario simplifica el análisis de datos para que puedas dedicar tiempo a extraer información de tus datos en lugar de luchar con una interfaz torpe. Admite una amplia variedad de datos de entrada, por lo que, a medida que su aplicación distribuida abarque más y diferentes tipos de servicios, puede seguir alimentando datos de registro y métricas en el sistema. Elastic Stack también admite búsquedas rápidas incluso en grandes conjuntos de datos, lo que permite incluso que las aplicaciones grandes registren datos detallados y puedan tener visibilidad sobre ellos de forma eficaz.

Logstash

El primer componente es Logstash. Esta herramienta se usa para recopilar información de registro de una gran variedad de orígenes diferentes. Por ejemplo, Logstash puede leer registros del disco y también recibir mensajes de bibliotecas de registro como Serilog. Logstash puede realizar algunos filtros y expansión básicos en los registros a medida que llegan. Por ejemplo, si los registros contienen direcciones IP, Logstash se puede configurar para

realizar una búsqueda geográfica y obtener un país o región o incluso una ciudad de origen para ese mensaje.

Serilog es una biblioteca de registro para lenguajes .NET, que permite el registro con parámetros. En lugar de generar un mensaje de registro de texto que inserte campos, los parámetros se mantienen separados. Esta biblioteca permite un filtrado y una búsqueda más inteligentes. Aparece una configuración de Serilog de ejemplo para escribir en Logstash en la figura 7-7.

```
C#

var log = new LoggerConfiguration()
          .WriteTo.Http("http://localhost:8080")
          .CreateLogger();
```

Figura 7-7. Configuración de Serilog para escribir información de registro directamente en logstash a través de HTTP

Logstash usaría una configuración como la que se muestra en la figura 7-8.

```

input {
    http {
        #default host 0.0.0.0:8080
        codec => json
    }
}

output {
    elasticsearch {
        hosts => "elasticsearch:9200"
        index=>"sales-%{+xxxx.ww}"
    }
}
```

Figura 7-8. Configuración de Logstash para consumir registros de Serilog

En escenarios en los que no se necesita una manipulación extensa del registro, hay una alternativa a Logstash conocida como Beats. Beats es una familia de herramientas con las que se puede recopilar una amplia variedad de datos desde registros a datos de red e información del tiempo de actividad. Muchas aplicaciones usarán Logstash y Beats.

Una vez recopilados los registros por Logstash, debe colocarlos en algún lugar. Aunque Logstash admite muchas salidas diferentes, una de las más emocionantes es Elasticsearch.

Elasticsearch

Elasticsearch es un motor de búsqueda eficaz que puede indexar los registros a medida que llegan. Agiliza las consultas en ejecución en los registros. Elasticsearch puede controlar grandes cantidades de registros y, en casos extremos, se puede escalar horizontalmente entre muchos nodos.

Los mensajes de registro que se han diseñado para contener parámetros o que han tenido parámetros divididos a través del procesamiento de Logstash, se pueden consultar directamente a medida que Elasticsearch conserva esta información.

Una consulta que busca las 10 páginas principales visitadas por `jill@example.com`, aparece en la figura 7-9.

JSON

```
"query": {
    "match": {
      "user": "jill@example.com"
    }
  },
  "aggregations": {
    "top_10_pages": {
      "terms": {
        "field": "page",
        "size": 10
      }
    }
  }
```

Figura 7-9. Una consulta de Elasticsearch para buscar las 10 páginas principales visitadas por un usuario

Visualización de información con paneles web de Kibana

El componente final de la pila es Kibana. Esta herramienta se usa para proporcionar visualizaciones interactivas en un panel web. Los paneles pueden ser diseñados incluso por los usuarios que no son técnicos. La mayoría de los datos que residen en el índice de Elasticsearch se pueden incluir en los paneles de Kibana. Los usuarios individuales pueden tener diferentes deseos de panel y Kibana permite esta personalización a través de permitir paneles específicos del usuario.

Instalación de Elastic Stack en Azure

La Elastic Stack se puede instalar en Azure de muchas maneras. Como siempre, es posible aprovisionar máquinas virtuales e instalar Elastic Stack directamente en ellas. Esta opción es preferida por algunos usuarios experimentados, ya que ofrece el mayor grado de personalización. Al implementarse en una infraestructura como servicio, se introduce una sobrecarga de administración significativa que obliga a los usuarios a tomar posesión de todas las tareas asociadas a la infraestructura como servicio como proteger máquinas y mantenerse al día con revisiones.

Una opción con menos sobrecarga es usar uno de los muchos contenedores de Docker en los que ya se ha configurado Elastic Stack. Estos contenedores se pueden agregar a un clúster de Kubernetes existente y ejecutarse junto con el código de la aplicación. El contenedor sebp/elk es un contenedor de Elastic Stack bien documentado y probado.

Otra opción es una oferta de ELK como servicio anunciada recientemente.

Referencias

- Instalación de Elastic Stack en Azure

Supervisión en Azure Kubernetes Services

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

El registro integrado en Kubernetes es primitivo. Sin embargo, hay algunas opciones excelentes para extraer los registros de Kubernetes a un lugar donde se pueden analizar correctamente. Si necesita supervisar los clústeres de AKS, la configuración de Elastic Stack para Kubernetes es una excelente solución.

Azure Monitor para contenedores

Azure Monitor para contenedores admite el consumo de registros desde no solo Kubernetes, sino también desde otros motores de orquestación, como DC/OS, Docker Swarm y Red Hat OpenShift.

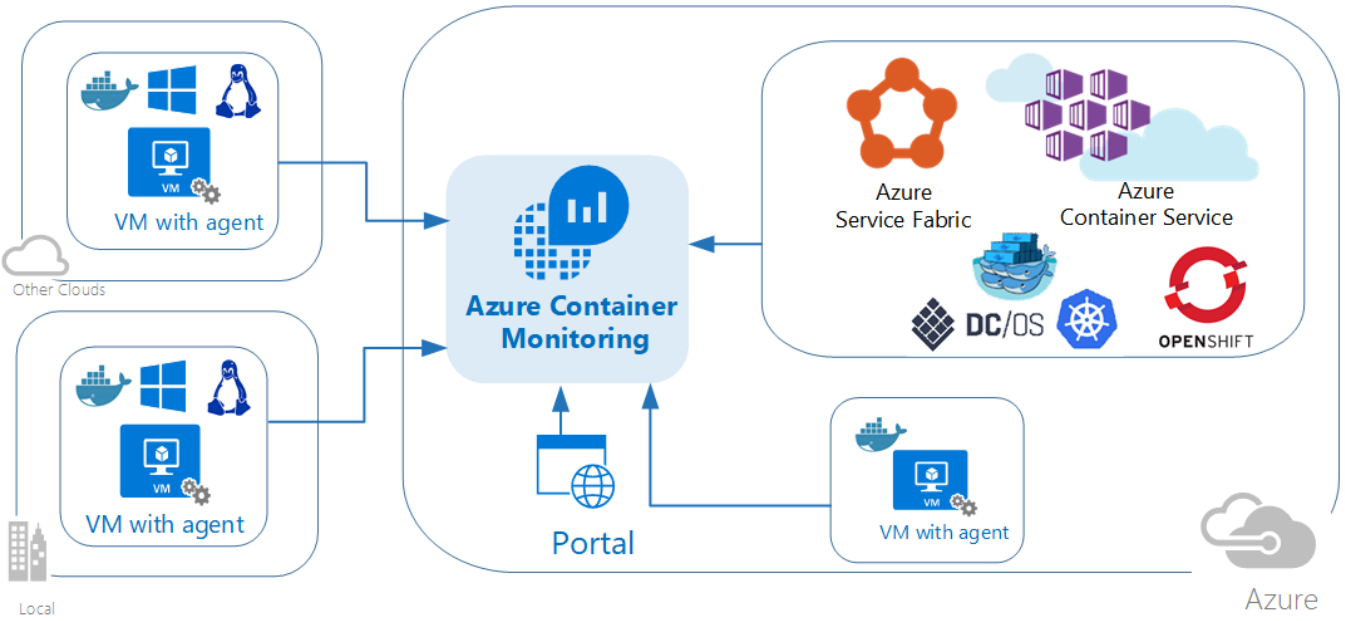


Figura 7-10. Consumo de registros de varios contenedores

Prometheus es una conocida solución de supervisión de métricas de código abierto. Forma parte de Cloud Native Compute Foundation. Normalmente, el uso de Prometheus requiere administrar un servidor de Prometheus con su propio almacén. Sin embargo, Azure Monitor para contenedores proporciona integración directa con puntos de conexión de métricas de Prometheus, por lo que no se requiere un servidor independiente.

La información de registro y métrica se recopila no solo de los contenedores que se ejecutan en el clúster, sino también de los propios hosts del clúster. Permite correlacionar la información del registro de los dos, lo que facilita mucho el seguimiento de un error.

La instalación de los recopiladores de registros difiere en los clústeres de Windows y Linux . Pero en ambos casos, la recopilación de registros se implementa como un DaemonSet de Kubernetes, lo que significa que el recopilador de registros se ejecuta como un contenedor en cada uno de los nodos.

Independientemente del orquestador o sistema operativo que ejecute el demonio de Azure Monitor, la información de registro se reenvía a las mismas herramientas de Azure Monitor con las que los usuarios están familiarizados. Este enfoque garantiza una experiencia paralela en entornos que combinan orígenes de registro diferentes, como un entorno híbrido de Kubernetes o Azure Functions.

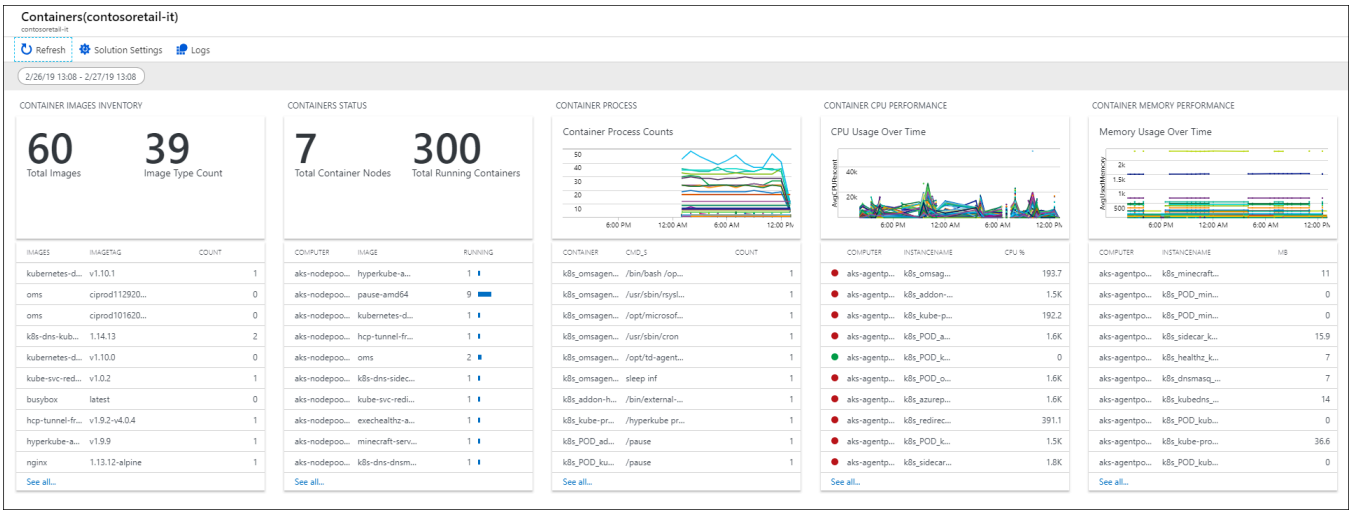


Figura 7-11. Panel de ejemplo que muestra información de registro y métricas de muchos contenedores en ejecución.

Log.Finalize()

El registro es una de las partes más pasadas por alto y, sin embargo, más importantes de la implementación de cualquier aplicación a escala. A medida que aumenta el tamaño y la complejidad de las aplicaciones, también aumenta la dificultad de depurarlas. Tener registros de alta calidad disponibles hace que la depuración sea mucho más fácil y la transforma de algo "casi imposible" en "una experiencia agradable".

Siguiente anterior

Azure Monitor

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Ningún otro proveedor de nube tiene tan madura una solución de supervisión de aplicaciones en la nube que la que se encuentra en Azure. Azure Monitor es un nombre paraguas para una colección de herramientas diseñadas para proporcionar visibilidad sobre el estado del sistema. Le ayuda a comprender cómo se realizan los servicios nativos de la nube e identifica proactivamente los problemas que les afectan. La figura 7-12 presenta una vista de alto nivel de Azure Monitor.

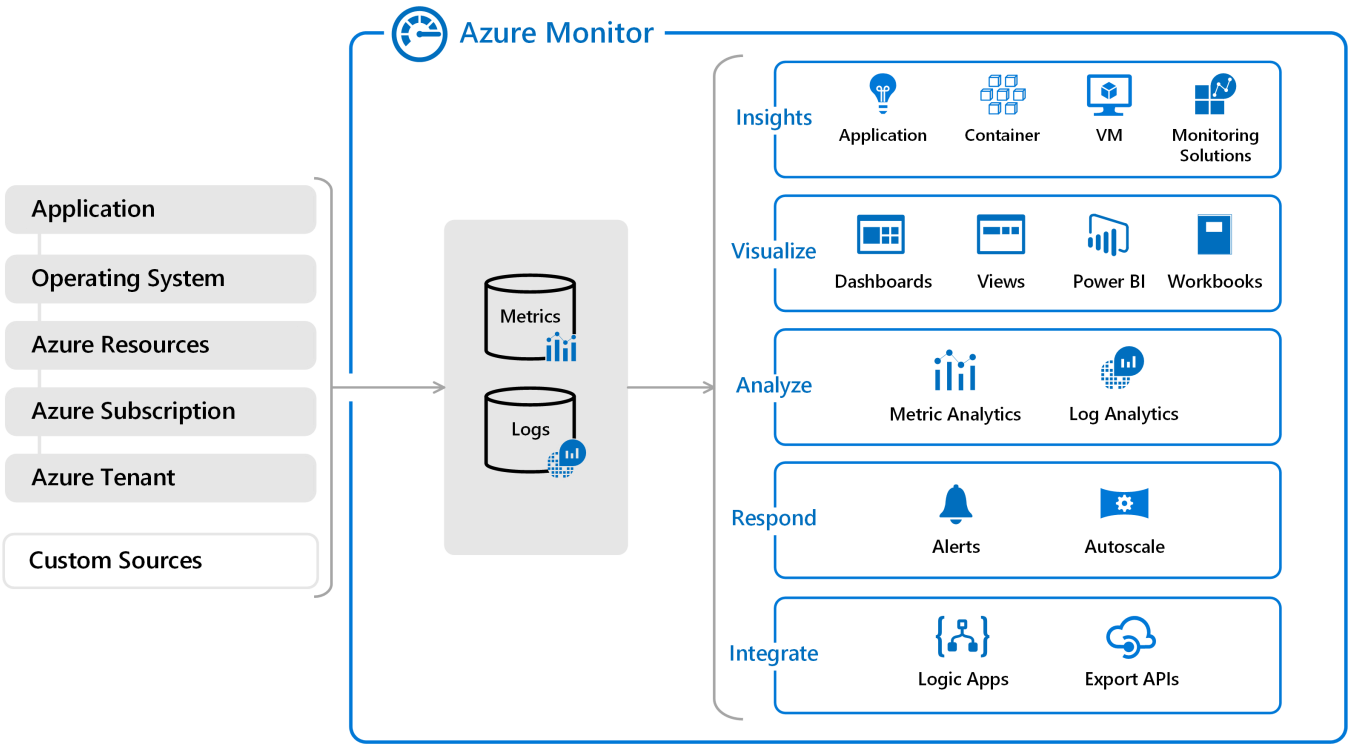


Figura 7-12. Vista general de Azure Monitor.

Recopilación de registros y métricas

El primer paso de cualquier solución de supervisión es recopilar tantos datos como sea posible. Cuantos más datos se recopilen, más profunda será la información. Tradicionalmente, los sistemas de instrumentación han sido difíciles. El Protocolo simple de administración de redes (SNMP) era el protocolo estándar de oro para recopilar información de nivel de máquina, pero requería una gran cantidad de conocimiento y configuración. Afortunadamente, gran parte de este trabajo duro se ha eliminado, ya que Azure Monitor recopila automáticamente las métricas más comunes.

Las métricas y los eventos de nivel de aplicación no son posibles instrumentar automáticamente porque son específicos de la aplicación que se está implementando. Para recopilar estas métricas, hay SDK y API disponibles para notificar directamente dicha información, como cuando un cliente se registra o completa un pedido. Las excepciones también se pueden capturar y notificar en Azure Monitor a través de Application Insights. Los SDK admiten la mayoría de los lenguajes que se encuentran en Aplicaciones nativas en la nube, como Go, Python, JavaScript y los lenguajes .NET.

El objetivo final de recopilar información sobre el estado de la aplicación es asegurarse de que los usuarios finales tengan una buena experiencia. ¿Qué mejor manera de saber si los usuarios experimentan problemas que hacer pruebas web externas? Estas pruebas pueden ser tan sencillas como hacer ping a su sitio web desde ubicaciones de todo el mundo o tan implicadas como tener agentes que inicien sesión en el sitio y simular acciones de usuario.

Creación de informes con los datos

Una vez recopilados los datos, se pueden manipular, resumir y trazar en gráficos, lo que permite a los usuarios ver al instante cuándo hay problemas. Estos gráficos se pueden recopilar en paneles o en libros, un informe de varias páginas diseñado para contar una historia sobre algún aspecto del sistema.

Ninguna aplicación moderna se completaría sin cierta inteligencia artificial ni aprendizaje automático. Para ello, los datos se pueden pasar a las distintas herramientas de aprendizaje automático de Azure para permitirle extraer tendencias e información que, de lo contrario, se ocultarían.

Application Insights proporciona un lenguaje de consulta eficaz (similar a SQL) denominado *Kusto* que puede consultar registros, resumirlos e incluso trazar gráficos. Por ejemplo, la consulta siguiente buscará todos los registros del mes de noviembre de 2007, los agrupará por estado y trazará los 10 primeros como gráfico circular.

Kusto

```
StormEvents
| where StartTime >= datetime(2007-11-01) and StartTime < datetime(2007-12-01)
| summarize count() by State
| top 10 by count_
| render piechart
```

En la figura 7-13 se muestran los resultados de esta consulta de Application Insights.

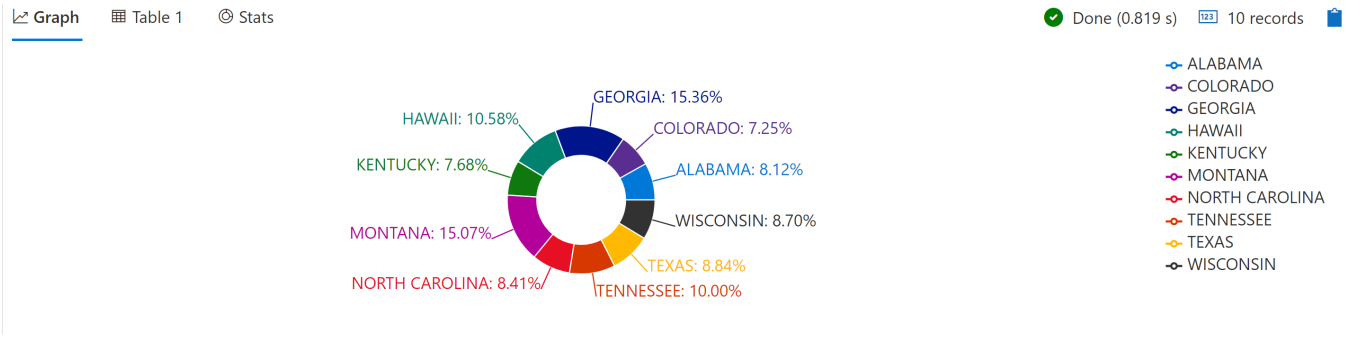


Figura 7-13. Resultados de la consulta de Application Insights.

Existe un área de juegos para experimentar con consultas de Kusto. La lectura de consultas de ejemplo también puede ser instructiva.

Tableros de control

Hay varias tecnologías de panel diferentes que se pueden usar para exponer la información de Azure Monitor. Quizás lo más sencillo es ejecutar consultas en Application Insights y trazar los

datos en un gráfico.

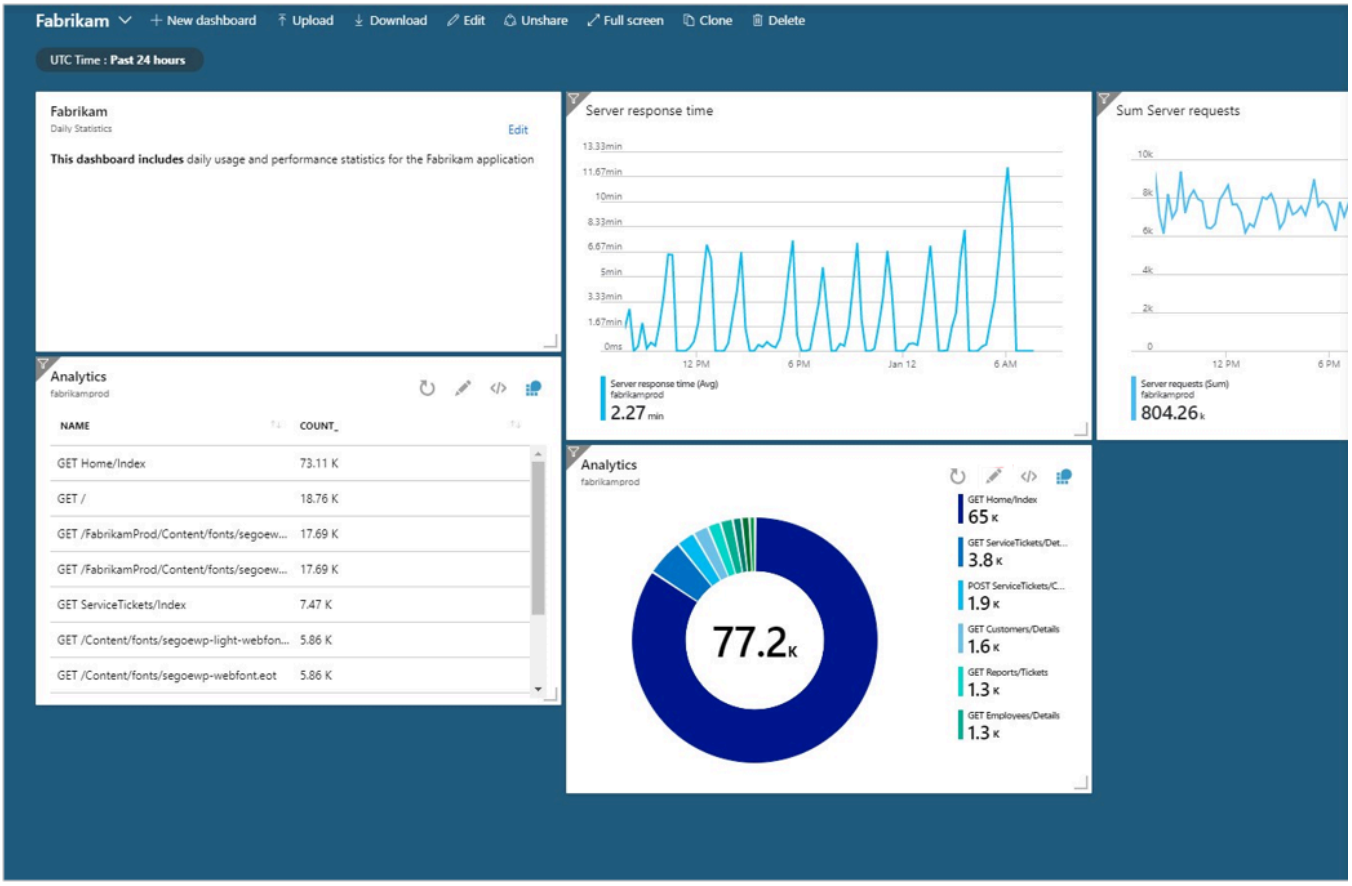


Figura 7-14. Ejemplo de gráficos de Application Insights insertados en el panel principal de Azure.

Estos gráficos se pueden incrustar directamente en el portal de Azure mediante el uso de la función de panel. Para los usuarios con requisitos más exactos, como la posibilidad de explorar en profundidad varios niveles de datos, los datos de Azure Monitor están disponibles para Power BI. Power BI es una herramienta de inteligencia empresarial líder en el sector que puede agregar datos de muchos orígenes de datos diferentes.

Figura 7-15. Un panel de Power BI de ejemplo.

Alertas

A veces, tener paneles de datos no es suficiente. Si nadie está despierto para monitorear los paneles, podrían pasar muchas horas antes de que se solucione un problema, o incluso se detecte. Para ello, Azure Monitor también proporciona una solución de alertas de primera categoría. Las alertas se pueden desencadenar mediante una amplia gama de condiciones, entre las que se incluyen:

- Valores de métrica
- Consultas de búsqueda de registros
- Eventos del registro de actividad
- Estado de la plataforma Azure subyacente
- Pruebas para la disponibilidad del sitio web

Cuando se desencadena, las alertas pueden realizar una amplia variedad de tareas. En el lado simple, las alertas solo pueden enviar una notificación por correo electrónico a una lista de correo o un mensaje de texto a una persona. Las alertas más implicadas pueden desencadenar un flujo de trabajo en una herramienta como PagerDuty, que es consciente de quién está en llamada para una aplicación determinada. Las alertas pueden desencadenar acciones en Microsoft Flow desbloqueando posibilidades casi ilimitadas para los flujos de trabajo.

A medida que se identifican las causas comunes de las alertas, las alertas se pueden mejorar con detalles sobre las causas comunes de las alertas y los pasos que se deben seguir para resolverlas. Las implementaciones de aplicaciones nativas de nube muy maduras pueden optar por iniciar tareas de recuperación automática, que realizan acciones como quitar nodos con errores de un conjunto de escalado o desencadenar una actividad de escalado automático. Finalmente, es posible que ya no sea necesario despertar al personal de guardia a las 2:00 AM para resolver un problema en el sitio web, ya que el sistema podrá ajustarse para compensar o al menos seguir funcionando hasta que alguien llegue al trabajo la mañana siguiente.

Azure Monitor aprovecha automáticamente el aprendizaje automático para comprender los parámetros operativos normales de las aplicaciones implementadas. Este enfoque permite detectar servicios que funcionan fuera de sus parámetros normales. Por ejemplo, el tráfico típico de la semana en el sitio puede ser de 10 000 solicitudes por minuto. Y luego, en una semana determinada, repentinamente el número de solicitudes alcanza un número muy inusual de 20 000 solicitudes por minuto. La detección inteligente observará esta desviación de la norma y desencadenará una alerta. Al mismo tiempo, el análisis de tendencias es suficientemente inteligente para evitar generar falsos positivos cuando se espera carga de tráfico.

Referencias

- Azure Monitor

Siguiente anterior

Identidad nativa de la nube

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

La mayoría de las aplicaciones de software necesitan tener algún conocimiento del usuario o proceso que los llama. El usuario o el proceso que interactúa con una aplicación se conoce como entidad de seguridad, y el proceso de autenticación y autorización de estas entidades de seguridad se conoce como administración de identidades o simplemente *identidad*. Las aplicaciones sencillas pueden incluir toda su administración de identidades dentro de la aplicación, pero este enfoque no se escala bien con muchas aplicaciones y muchos tipos de entidades de seguridad. Windows admite el uso de Active Directory para proporcionar autenticación y autorización centralizadas.

Aunque esta solución es eficaz dentro de las redes corporativas, no está diseñada para su uso por parte de usuarios o aplicaciones que están fuera del dominio de AD. Con el crecimiento de las aplicaciones basadas en Internet y el aumento de las aplicaciones nativas de la nube, los modelos de seguridad han evolucionado.

En el modelo de identidad nativa de la nube actual, se supone que la arquitectura se distribuye. Las aplicaciones se pueden implementar en cualquier lugar y pueden comunicarse con otras aplicaciones en cualquier lugar. Los clientes pueden comunicarse con estas aplicaciones desde cualquier lugar y, de hecho, los clientes pueden constar de cualquier combinación de plataformas y dispositivos. Las soluciones de identidad nativa de la nube usan estándares abiertos para lograr un acceso seguro a las aplicaciones de los clientes. Estos clientes van desde usuarios humanos en equipos o teléfonos, a otras aplicaciones hospedadas en cualquier

lugar en línea, y a decodificadores y dispositivos IOT que ejecutan cualquier plataforma de software en cualquier lugar del mundo.

Las soluciones de identidad nativas de nube modernas suelen usar tokens de acceso emitidos por un servicio o servidor de token seguro (STS) a una entidad de seguridad una vez determinada su identidad. El token de acceso, normalmente un JSON Web Token (JWT), incluye *notificaciones* sobre la entidad de seguridad. Estas notificaciones incluirán como mínimo la identidad del usuario, pero también otras notificaciones que las aplicaciones pueden usar para determinar el nivel de acceso que conceder a una entidad de seguridad.

Normalmente, el STS solo es responsable de autenticar al principal. La determinación de su nivel de acceso a los recursos queda a otras partes de la aplicación.

Referencias

- Plataforma de identidad de Microsoft

Siguiente anterior

Autenticación y autorización en aplicaciones nativas de la nube

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

La autenticación es el proceso de determinar la identidad de una entidad de seguridad. *La autorización* es el acto de conceder permiso a un principal autenticado para realizar una acción o acceder a un recurso. A veces, la autenticación se abrevia a `AuthN` y la autorización se acorta a `AuthZ`. Las aplicaciones nativas de nube deben confiar en protocolos abiertos basados en HTTP para autenticar entidades de seguridad, ya que tanto los clientes como las aplicaciones podrían ejecutarse en cualquier lugar del mundo en cualquier plataforma o dispositivo. El único factor común es HTTP.

Muchas organizaciones siguen confiando en servicios de autenticación local como Servicios de federación de Active Directory (ADFS). Aunque este enfoque ha servido tradicionalmente a las organizaciones bien para las necesidades de autenticación local, las aplicaciones nativas de la nube se benefician de sistemas diseñados específicamente para la nube. Un reciente aviso del Centro nacional de seguridad cibernética (NCSC) del Reino Unido de 2019 indica que "las organizaciones que usan Azure AD como origen de autenticación principal realmente reducirán su riesgo en comparación con ADFS". Algunas razones que se describen en este análisis incluyen:

- Acceso a un conjunto completo de tecnologías de protección de credenciales de Microsoft.
- La mayoría de las organizaciones ya se basan en Azure AD en cierta medida.

- El hash doble de hashes NTLM garantiza que no se permitan credenciales que funcionen en Active Directory local.

Referencias

- Conceptos básicos sobre autenticación
- Tokens de acceso y notificaciones
- Puede ser el momento de abandonar los servicios de autenticación locales.

Siguiente anterior

Azure Active Directory

17/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Microsoft Azure Active Directory (Azure AD) ofrece administración de identidades y acceso como servicio. Los clientes lo usan para configurar y mantener quiénes son los usuarios, qué información almacenar sobre ellos, quién puede acceder a esa información, quién puede administrarla y qué aplicaciones pueden acceder a ella. AAD puede autenticar a los usuarios para que las aplicaciones configuradas lo usen, lo que proporciona una experiencia de inicio de sesión único (SSO). Se puede usar por sí mismo o integrarse con Windows AD que se ejecuta en el entorno local.

Azure AD se ha creado para la nube. Es realmente una solución de identidad nativa de la nube que usa graph API basada en REST y sintaxis de OData para consultas, a diferencia de Windows AD, que usa LDAP. Active Directory local puede sincronizar atributos de usuario en la nube mediante Identity Sync Services, lo que permite que toda la autenticación se realice en la nube mediante Azure AD. Como alternativa, la autenticación se puede configurar mediante Connect para redirigirla al Active Directory local a través de ADFS, y que sea completada in situ por Windows AD.

Azure AD admite pantallas de inicio de sesión con marca de empresa, autenticación multifactor y servidores proxy de aplicación basados en la nube que se usan para proporcionar SSO para aplicaciones hospedadas en el entorno local. Ofrece diferentes tipos de funcionalidades de informes de seguridad y alertas.

Referencias

- Plataforma de identidad de Microsoft

Siguiente anterior

IdentityServer para aplicaciones nativas de nube

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

Duende IdentityServer es un marco para compilar openID Connect (OIDC) y un servidor de autenticación compatible con estándares de OAuth 2.x mediante ASP.NET Core.

Está diseñado para proporcionar una manera común de autenticar las solicitudes a todas las aplicaciones, tanto si son puntos de conexión web, nativos, móviles o de API. IdentityServer se puede usar para implementar single Sign-On (SSO) para varias aplicaciones y tipos de aplicación. Se puede usar para autenticar usuarios reales a través de formularios de inicio de sesión y interfaces de usuario similares, así como la autenticación basada en servicios que normalmente implica la emisión, comprobación y renovación de tokens sin ninguna interfaz de usuario. También puede actuar como puerta de enlace de federación para unificar proveedores de autenticación.

IdentityServer está diseñado para ser una solución personalizable. Cada instancia suele personalizarse para adaptarse a una organización individual o a las necesidades de un conjunto de aplicaciones.

Escenarios comunes de aplicaciones web

Normalmente, las aplicaciones deben admitir algunos o todos los escenarios siguientes:

- Usuarios humanos que acceden a aplicaciones web con un explorador.

- Usuarios humanos que acceden a las API web de back-end desde aplicaciones basadas en explorador.
- Usuarios humanos en clientes móviles o nativos que acceden a las API web de back-end.
- Otras aplicaciones que acceden a las API web de back-end (sin un usuario activo o una interfaz de usuario).
- Es posible que cualquier aplicación tenga que interactuar con otras API web mediante su propia identidad o delegarla en la identidad del usuario.

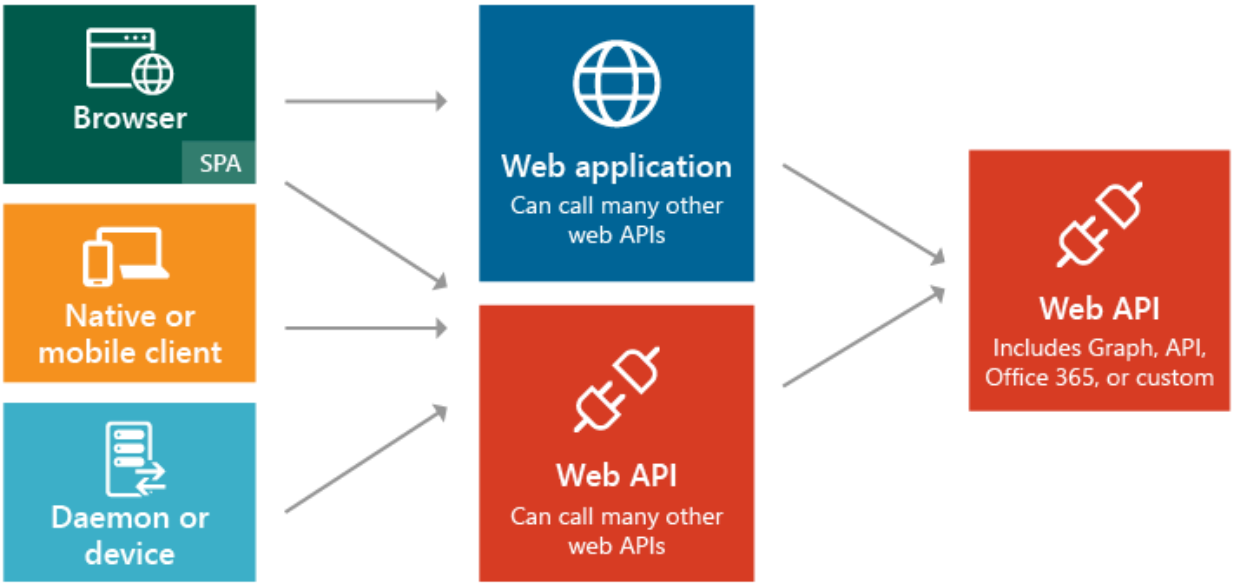


Figura 8-1. Tipos y escenarios de aplicación.

En cada uno de estos escenarios, la funcionalidad expuesta debe protegerse frente al uso no autorizado. Como mínimo, esto suele requerir la autenticación del usuario o la entidad de seguridad que realiza una solicitud para un recurso. Esta autenticación puede usar uno de varios protocolos comunes, como SAML2p, WS-Fed o OpenID Connect. La comunicación con las API suele usar el protocolo OAuth 2 y su compatibilidad con tokens de seguridad. Separar estos problemas críticos de seguridad transversales y sus detalles de implementación de las propias aplicaciones garantiza la coherencia y mejora la seguridad y el mantenimiento. La subcontratación de estos problemas a un producto dedicado, como IdentityServer, ayuda a eliminar la necesidad de que cada aplicación resuelva estos problemas por sí misma.

IdentityServer proporciona middleware que se ejecuta dentro de una aplicación ASP.NET Core y agrega compatibilidad con OpenID Connect y OAuth 2.x (consulte las especificaciones admitidas). Con IdentityServer, las organizaciones pueden crear su propia aplicación ASP.NET Core mediante el middleware IdentityServer para actuar como servidor de autorización para todos sus protocolos de seguridad basados en tokens. El middleware IdentityServer expone puntos de conexión para admitir la funcionalidad estándar, entre las que se incluyen:

- Autorización (autenticación del usuario final)

- Token (solicite un token mediante programación)
- Detección (metadatos sobre el servidor)
- Información de usuario (obtener información de usuario con un token de acceso válido)
- Autorización del dispositivo (que se usa para iniciar la autorización de flujo de dispositivo)
- Introspección (validación de tokens)
- Revocación (anulación de tokens)
- Finalizar sesión (desencadenar cierre de sesión único en todas las aplicaciones)
- Solicitudes de autorización enviadas (para un proceso más seguro de autenticación)

Empezar

IdentityServer está disponible:

- Con una licencia comunitaria, que le permite usar el IdentityServer gratis para pequeñas empresas y sin fines de lucro (se aplican condiciones)
- La versión de pago, que te permite usar IdentityServer en un escenario comercial

Para obtener información más detallada sobre los precios, consulte la página de precios oficial.

Puede agregarlo a las aplicaciones mediante sus paquetes NuGet. El paquete principal es IdentityServer, que se ha descargado más de cuatro millones de veces. El paquete base no incluye ningún código de interfaz de usuario y solo admite la configuración en memoria. Para usarlo con una base de datos, también querrá de un proveedor de datos como Duende.IdentityServer.Storage, que usa Entity Framework Core para almacenar la configuración y los datos operativos de IdentityServer. En el caso de la interfaz de usuario, puede copiar archivos desde el repositorio de ejemplos en la aplicación ASP.NET Core MVC para agregar compatibilidad con el inicio de sesión y cierre de sesión mediante el middleware IdentityServer.

Configuración

IdentityServer admite diferentes tipos de protocolos y proveedores de autenticación social que se pueden configurar como parte de cada instalación personalizada. Normalmente, esto se hace en la clase de `Program` la aplicación ASP.NET Core. La configuración implica especificar los protocolos admitidos y las rutas de acceso a los servidores y puntos de conexión que se usarán. En la figura 8-2 se muestra una configuración de ejemplo tomada de la Guía rápida de IdentityServer para el proyecto de aplicaciones ASP.NET Core:

```
C#
```

```
// some details omitted
builder.Services.AddIdentityServer();

builder.Services.AddAuthentication(options =>
    {
        options.DefaultScheme = "Cookies";
        options.DefaultChallengeScheme = "oidc";
    })
    .AddCookie("Cookies")
    .AddGoogle("Google", options =>
    {
        options.SignInScheme =
IdentityServerConstants.ExternalCookieAuthenticationScheme;

        options.ClientId = "<insert here>";
        options.ClientSecret = "<insert here>";
    })
    .AddOpenIdConnect("oidc", options =>
    {
        options.Authority = "https://localhost:5001";

        options.ClientId = "web";
        options.ClientSecret = "secret";
        options.ResponseType = "code";

        options.Scope.Clear();
        options.Scope.Add("openid");
        options.Scope.Add("profile");

        options.MapInboundClaims = false; // Don't rename claim types

        options.SaveTokens = true;
    });
}
```

Figura 8-2. Configuración de IdentityServer.

Clientes de JavaScript

Muchas aplicaciones nativas de nube usan API del lado servidor y aplicaciones de página única (SPA) de cliente enriquecidas en el front-end, por ejemplo, mediante React, Angular o Blazor WebAssembly. El patrón backend para frontend (BFF) se usa para este tipo de clientes, lo que permite mantener los tokens fuera del alcance del explorador. Este patrón sigue la especificación OAuth 2.0 de IETF para Browser-Based Applications.

Referencias

- Documentación de IdentityServer

- tipos de aplicación
- Backend para frontend

siguiente

Last updated on 23/12/2025

Seguridad nativa de la nube

17/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

No pasa un día en el que las noticias no contienen una historia sobre una empresa que se hackea o pierden los datos de sus clientes. Incluso los países o regiones no son inmunes a los problemas creados al tratar la seguridad como una idea posterior. Durante años, las empresas han tratado la seguridad de los datos de los clientes y, de hecho, sus redes completas como algo de "agradable para tener". Los servidores de Windows se dejaron sin actualizar, las versiones antiguas de PHP se seguían ejecutando y las bases de datos de MongoDB se dejaban abiertas al mundo.

Sin embargo, están comenzando a aparecer consecuencias reales por no mantener una mentalidad de seguridad al crear e implementar aplicaciones. Muchas empresas aprendieron por las malas lo que puede ocurrir cuando los servidores y equipos de escritorio no se actualizan durante el brote de 2017 de NotPetya. El costo de estos ataques ha llegado fácilmente a los miles de millones de dólares, con algunas estimaciones que ponen las pérdidas de este único ataque a 10 mil millones de dólares estadounidenses.

Incluso los gobiernos no son inmunes a incidentes de piratería. La ciudad de Baltimore fue tomada como rehén por criminales, lo que hizo imposible que los ciudadanos pagaran sus facturas o usaran los servicios de la ciudad.

También ha habido un aumento de la legislación que exige ciertas protecciones de datos para los datos personales. En Europa, el RGPD ha entrado en vigor durante más de un año y, más recientemente, California pasó su propia versión denominada CCDA, que entra en vigor el 1 de

enero de 2020. Las multas en virtud del RGPD pueden ser tan penales como para poner a las empresas fuera de negocio. Google ya ha sido multado con 50 millones de euros por infracciones, pero eso es solo una gota en el océano en comparación con las posibles multas.

En resumen, la seguridad es un negocio serio.

Siguiente anterior

Seguridad de Azure para aplicaciones nativas en la nube

Artículo • 27/05/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Las aplicaciones nativas de la nube pueden ser más fáciles y difíciles de proteger que las aplicaciones tradicionales. A la desventaja, debe proteger aplicaciones más pequeñas y dedicar más energía a crear la infraestructura de seguridad. La naturaleza heterogénea de los lenguajes de programación y los estilos en la mayoría de las implementaciones de servicios también significa que debe prestar más atención a los boletines de seguridad de muchos proveedores diferentes.

Por otro lado, los servicios más pequeños, cada uno con su propio almacén de datos, limitan el ámbito de un ataque. Si un atacante pone en peligro un sistema, es probable que sea más difícil que el atacante realice el salto a otro sistema que en una aplicación monolítica. Los límites de proceso son límites seguros. Además, si se expone una copia de seguridad de base de datos, el daño es más limitado, ya que esa base de datos solo contiene un subconjunto de datos y es poco probable que contenga datos personales.

Modelado de amenazas

Independientemente de si las ventajas superan las desventajas de las aplicaciones nativas de la nube, se debe seguir la misma mentalidad de seguridad holística. La seguridad y el enfoque a la seguridad deben formar parte de cada paso del proceso de desarrollo y operaciones. Al planear una aplicación, haga preguntas como:

- ¿Cuál sería el impacto de estos datos que se pierden?
- ¿Cómo podemos limitar el daño de los datos incorrectos que se insertan en este servicio?
- ¿Quién debe tener acceso a estos datos?
- ¿Existen directivas de auditoría en torno al proceso de desarrollo y versión?

Todas estas preguntas forman parte de un proceso denominado modelado de amenazas. Este proceso intenta responder a la pregunta de qué amenazas hay en el sistema, la probabilidad de que las amenazas sean y el posible daño de ellas.

Una vez establecida la lista de amenazas, debe decidir si merece la pena mitigarlas. A veces, una amenaza es tan poco probable y costosa de planear para que no valga la pena gastar energía en ella. Por ejemplo, algunos actores de nivel de estado podrían insertar cambios en el diseño de un proceso que usan millones de dispositivos. Ahora, en lugar de ejecutar un determinado fragmento de código en el anillo 3, ese código se ejecuta en el anillo 0. Este proceso permite una vulnerabilidad de seguridad que puede omitir el hipervisor y ejecutar el código de ataque en las máquinas sin sistema operativo, lo que permite ataques en todas las máquinas virtuales que se ejecutan en ese hardware.

Los procesadores modificados son difíciles de detectar sin un microscopio y conocimientos avanzados sobre el diseño de silicio de ese procesador. Es poco probable que este escenario se produzca y sea costoso mitigarlo, por lo que probablemente ningún modelo de amenazas recomendaría crear protección contra vulnerabilidades de seguridad.

Las amenazas más probables, como los controles de acceso rotos que permiten ataques de incremento (reemplazando `Id=2` por `Id=3` en la dirección URL) o la inyección SQL, son más atractivas para desarrollar protecciones. Las mitigaciones de estas amenazas son bastante razonables para crear y evitar agujeros de seguridad embarazosos que manchan la reputación de la empresa.

Principio de privilegio mínimo

Una de las ideas fundamentales en la seguridad informática es el principio de privilegios mínimos (POLP). En realidad es una idea fundamental en la mayoría de las formas de seguridad, ya sea digital o física. En resumen, el principio es que cualquier usuario o proceso debe tener el menor número de derechos posibles para ejecutar su tarea.

Por ejemplo, piense en los tellers de un banco: acceder a la caja fuerte es una actividad poco común. Por lo tanto, el cajero promedio no puede abrir la caja fuerte. Para obtener acceso, deben escalar su solicitud a través de un administrador bancario, que realiza comprobaciones de seguridad adicionales.

En un sistema informático, un ejemplo fantástico es los derechos de un usuario que se conecta a una base de datos. En muchos casos, hay una única cuenta de usuario que se usa para compilar la estructura de la base de datos y ejecutar la aplicación. Excepto en casos extremos, la cuenta que ejecuta la aplicación no necesita la capacidad de actualizar la información del esquema. Debe haber varias cuentas que proporcionen distintos niveles de privilegios. La aplicación solo debe usar el nivel de permiso que concede acceso de lectura y escritura a los datos de las tablas. Este tipo de protección eliminaría los ataques destinados a quitar tablas de base de datos o introducir desencadenadores malintencionados.

Casi todas las partes de la creación de una aplicación nativa de la nube pueden beneficiarse de recordar el principio de privilegios mínimos. Puede encontrarlo en acción al configurar firewalls, grupos de seguridad de red, roles y ámbitos en el control de acceso basado en roles (RBAC).

Pruebas de penetración

A medida que las aplicaciones se complican más el número de vectores de ataque aumenta a una velocidad alarmante. El modelado de amenazas es defectuoso en que tiende a ser ejecutado por las mismas personas que compilan el sistema. De la misma manera que muchos desarrolladores tienen problemas para visualizar las interacciones del usuario y, a continuación, crear interfaces de usuario no utilizables, la mayoría de los desarrolladores tienen dificultades para ver cada vector de ataque. También es posible que los desarrolladores que compilan el sistema no estén bien versados en metodologías de ataque y pierdan algo crucial.

Las pruebas de penetración o las "pruebas de lápiz" implican incorporar actores externos para intentar atacar el sistema. Estos atacantes pueden ser una empresa de consultoría externa u otros desarrolladores con buenos conocimientos de seguridad de otra parte de la empresa. Se les da carta blanca para intentar subvertir el sistema. Con frecuencia, encontrarán amplios agujeros de seguridad que deben revisarse. A veces, el vector de ataque será algo totalmente inesperado, como aprovechar un ataque de suplantación de identidad contra el CEO.

Azure en sí está experimentando constantemente ataques de un equipo de hackers dentro de Microsoft. A lo largo de los años, han sido los primeros en encontrar docenas de vectores de ataque potencialmente catastróficos, cerrandolos antes de que se puedan aprovechar externamente. Cuanto más tentador sea un objetivo, más probable es que los actores externos intenten aprovecharlo y que haya pocos destinos en el mundo más tentadores que Azure.

Monitorización

Si un atacante intenta penetrar una aplicación, debería haber alguna advertencia. Con frecuencia, los ataques se pueden detectar examinando los registros de los servicios. Los

ataques dejan señales reveladoras que se pueden detectar antes de que tengan éxito. Por ejemplo, un atacante que intenta adivinar una contraseña realizará muchas solicitudes a un sistema de inicio de sesión. La supervisión en torno al sistema de inicio de sesión puede detectar patrones extraños que están fuera de línea con el patrón de acceso típico. Esta supervisión se puede convertir en una alerta que, a su vez, puede alertar a una persona de operaciones para activar algún tipo de contramedida. Un sistema de supervisión muy maduro puede incluso tomar medidas en función de estas desviaciones agregando reglas de forma proactiva para bloquear las solicitudes o limitar las respuestas.

Protección de la compilación

Un lugar donde a menudo se pasa por alto la seguridad está en torno al proceso de compilación. No solo se deben ejecutar comprobaciones de seguridad durante la compilación, como el análisis de código inseguro o las credenciales registradas, sino que la propia compilación debe ser segura. Si el servidor de compilación está en peligro, proporciona un vector fantástico para introducir código arbitrario en el producto.

Imagine que un atacante busca robar las contraseñas de las personas que inician sesión en una aplicación web. Podrían introducir una etapa de construcción que modifique el código extraído para redirigir cualquier solicitud de inicio de sesión hacia otro servidor. La próxima vez que el código pase por la compilación, se actualiza de forma silenciosa. El examen de vulnerabilidades de código fuente no detectará esta vulnerabilidad, ya que se ejecuta antes de la compilación. De igual manera, nadie se dará cuenta en una revisión de código porque las etapas de construcción residen en el servidor de compilación. El código vulnerado irá a producción donde puede recopilar contraseñas. Probablemente no haya ningún registro de auditoría de los cambios en el proceso de construcción, o al menos nadie está supervisando dicha auditoría.

Este escenario es un ejemplo perfecto de un objetivo aparentemente de bajo valor que se puede usar para penetrar en el sistema. Una vez que un atacante infringe el perímetro del sistema, puede empezar a trabajar en la búsqueda de formas de elevar sus permisos hasta el punto de que puedan causar daños reales en cualquier lugar que les guste.

Creación de código seguro

.NET Framework ya es un marco bastante seguro. Evita algunos de los riesgos del código no administrado, como salirse de los límites de las matrices. El trabajo se realiza activamente para corregir los agujeros de seguridad a medida que se detectan. Incluso hay un programa de recompensas por errores que paga a los investigadores para encontrar problemas en el marco y notificarlos en lugar de aprovecharlos.

Hay muchas maneras de proteger el código de .NET. Las instrucciones siguientes, como las directrices de codificación segura para el artículo de .NET , son un paso razonable que se debe seguir para asegurarse de que el código es seguro desde cero. El OWASP top 10 es otra guía inestimable para crear código seguro.

El proceso de compilación es un buen lugar para poner herramientas de examen para detectar problemas en el código fuente antes de convertirlos en producción. La mayoría de los proyectos tienen dependencias en algunos otros paquetes. Una herramienta que puede buscar paquetes obsoletos detectará problemas en una compilación nocturna. Incluso al compilar imágenes de Docker, resulta útil comprobar y asegurarse de que la imagen base no tiene vulnerabilidades conocidas. Otra cosa que hay que comprobar es que nadie ha comprobado accidentalmente las credenciales.

Seguridad integrada

Azure está diseñado para equilibrar la facilidad de uso y la seguridad de la mayoría de los usuarios. Los distintos usuarios tendrán requisitos de seguridad diferentes, por lo que deben ajustar su enfoque a la seguridad en la nube. Microsoft publica una gran cantidad de información de seguridad en el Centro de confianza . Este recurso debe ser la primera parada para aquellos profesionales interesados en comprender cómo funcionan las tecnologías integradas de mitigación de ataques.

En Azure Portal, Azure Advisor es un sistema que examina constantemente un entorno y realiza recomendaciones. Algunas de estas recomendaciones están diseñadas para ahorrar dinero a los usuarios, pero otras están diseñadas para identificar configuraciones potencialmente no seguras, como tener un contenedor de almacenamiento abierto al mundo y no protegidos por una red virtual.

Infraestructura de red de Azure

En un entorno de implementación local, una gran cantidad de energía se dedica a configurar redes. La configuración de enrutadores, conmutadores y cosas por el estilo es complicada. Las redes permiten que determinados recursos hablen con otros recursos y eviten el acceso en algunos casos. Una regla de red frecuente es restringir el acceso al entorno de producción desde el entorno de desarrollo en caso de que un fragmento de código parcialmente desarrollado falle y borre una cantidad de datos.

De fábrica, la mayoría de los recursos de PaaS de Azure solo tienen la configuración de red más básica y permisiva. Por ejemplo, cualquier usuario de Internet puede acceder a un servicio de aplicaciones. Las nuevas instancias de SQL Server suelen estar restringidas, de modo que las partes externas no puedan acceder a ellas, pero se permiten los intervalos de direcciones IP

usados por Azure en sí. Por lo tanto, aunque SQL Server está protegido frente a amenazas externas, un atacante solo necesita configurar una cabeza de puente de Azure desde donde pueden iniciar ataques contra todas las instancias de SQL en Azure.

Afortunadamente, la mayoría de los recursos de Azure se pueden colocar en una instancia de Azure Virtual Network que permite un control de acceso específico. De forma similar a la forma en que las redes locales establecen redes privadas que están protegidas del mundo más amplio, las redes virtuales son islas de direcciones IP privadas que se encuentran dentro de la red de Azure.

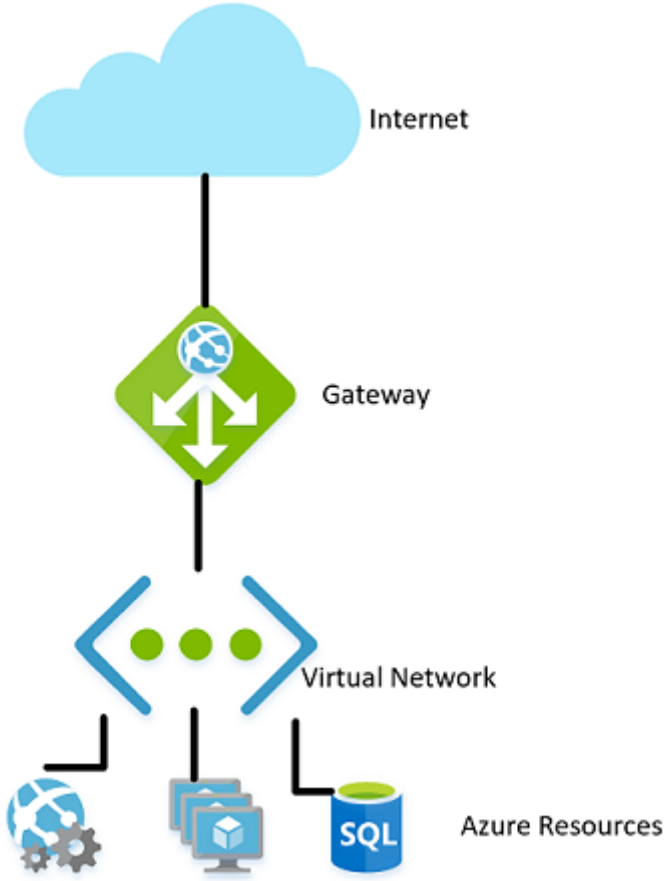


Figura 9-1. Una red virtual en Azure.

De la misma manera que las redes locales tienen un firewall que rige el acceso a la red, puede establecer un firewall similar en el límite de la red virtual. De forma predeterminada, todos los recursos de una red virtual todavía pueden comunicarse con Internet. Solo son conexiones entrantes que requieren alguna forma de excepción de firewall explícita.

Con la red establecida, los recursos internos, como las cuentas de almacenamiento, solo se pueden configurar para permitir el acceso por los recursos que también están en la red virtual. Este firewall proporciona un nivel adicional de seguridad, si se filtran las claves de esa cuenta de almacenamiento, los atacantes no podrían conectarse a ella para aprovechar las claves filtradas. Este escenario es otro ejemplo del principio de privilegios mínimos.

Los nodos de un clúster de Azure Kubernetes pueden participar en una red virtual igual que otros recursos que son más nativos de Azure. Esta funcionalidad se denomina Interfaz de red de contenedor de Azure. De hecho, asigna una subred dentro de la red virtual en la que se asignan las máquinas virtuales y las imágenes de contenedor.

Continuando con la ilustración del principio de privilegios mínimos, no todos los recursos de una Red Virtual necesitan comunicarse con cada uno de los demás recursos. Por ejemplo, en una aplicación que proporciona una API web a través de una cuenta de almacenamiento y una base de datos SQL, es poco probable que la base de datos y la cuenta de almacenamiento necesiten comunicarse entre sí. Cualquier uso compartido de datos entre ellos pasará por la aplicación web. Por lo tanto, se podría usar un grupo de seguridad de red (NSG) para denegar el tráfico entre los dos servicios.

Una directiva de prohibir la comunicación entre recursos puede ser molesta para implementar, especialmente cuando se proviene de un contexto de Azure sin restricciones de tráfico. En otras nubes, el concepto de grupos de seguridad de red es mucho más frecuente. Por ejemplo, la directiva predeterminada en AWS es que los recursos no se pueden comunicar entre sí hasta que las reglas de un NSG lo habiliten. Aunque es más lento desarrollar esto, un entorno más restrictivo proporciona un valor predeterminado más seguro. El uso de procedimientos adecuados de DevOps, especialmente el uso de Azure Resource Manager o Terraform para administrar permisos puede facilitar el control de las reglas.

Las redes virtuales también pueden ser útiles al configurar la comunicación entre recursos locales y en la nube. Se puede usar una red privada virtual para conectar sin problemas las dos redes. Este enfoque permite ejecutar una red virtual sin ningún tipo de puerta de enlace para escenarios en los que todos los usuarios están en el sitio. Hay una serie de tecnologías que se pueden usar para establecer esta red. Lo más sencillo es usar una VPN de sitio a sitio que se puede establecer entre muchos enrutadores y Azure. El tráfico se cifra y se tuneliza a través de Internet con el mismo costo por byte que cualquier otro tráfico. En escenarios en los que es deseable más ancho de banda o más seguridad, Azure ofrece un servicio denominado ExpressRoute que usa un circuito privado entre una red local y Azure. Es más costoso y difícil de establecer, pero también más seguro.

Control de acceso basado en rol para restringir el acceso a los recursos de Azure

RBAC es un sistema que proporciona una identidad a las aplicaciones que se ejecutan en Azure. Las aplicaciones pueden acceder a los recursos mediante esta identidad en lugar de o además de usar claves o contraseñas.

Entidades de seguridad

El primer componente de RBAC es una entidad de seguridad. Una entidad de seguridad puede ser un usuario, un grupo, una entidad de servicio o una identidad administrada.

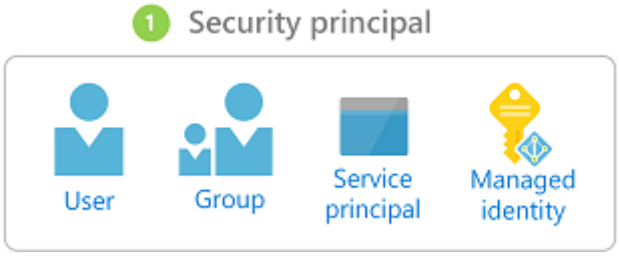


Figura 9-2. Diferentes tipos de entidades de seguridad.

- Usuario: cualquier usuario que tenga una cuenta en Azure Active Directory es un usuario.
- Grupo: colección de usuarios de Azure Active Directory. Como miembro de un grupo, un usuario asume los roles de ese grupo además de los suyos propios.
- Entidad de servicio: una identidad de seguridad bajo la cual se ejecutan servicios o aplicaciones.
- Identidad administrada: una identidad de Azure Active Directory administrada por Azure. Las identidades administradas se suelen usar al desarrollar aplicaciones en la nube que administran las credenciales para autenticarse en los servicios de Azure.

La entidad de seguridad se puede aplicar a casi cualquier recurso. Este aspecto significa que es posible asignar una entidad de seguridad a un contenedor que se ejecuta en Azure Kubernetes, lo que le permite acceder a secretos almacenados en Key Vault. Una función de Azure podría asumir un permiso que le permita comunicarse con una instancia de Active Directory para validar un JWT para un usuario que llama. Una vez habilitados los servicios con una entidad de servicio, sus permisos se pueden administrar de forma granular mediante roles y ámbitos.

Funciones

Una entidad de seguridad puede asumir muchos roles o, con una analogía más sartorial, llevar muchos sombreros. Cada rol define una serie de permisos, como "Leer mensajes del punto de conexión de Azure Service Bus". El conjunto de permisos efectivo de una entidad de seguridad es la combinación de todos los permisos asignados a todos los roles que tiene una entidad de seguridad. Azure tiene un gran número de roles integrados y los usuarios pueden definir sus propios roles.

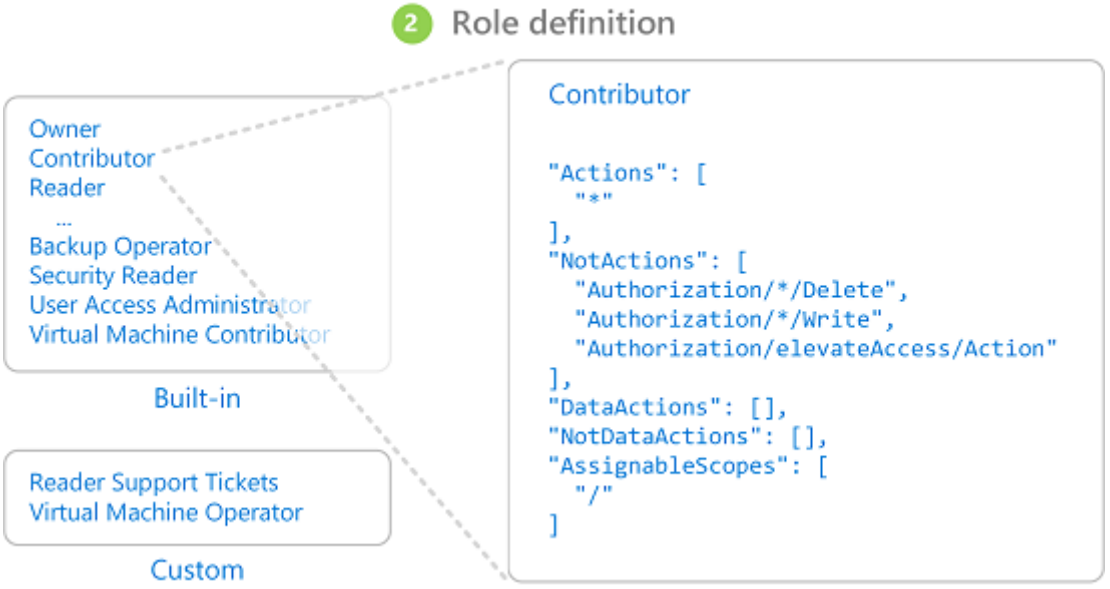


Figura 9-3. Definiciones de roles de RBAC.

Azure tiene integrados varios roles de alto nivel, como propietario, colaborador, lector y administrador de cuentas de usuario. Con el rol de Propietario, una entidad de seguridad puede acceder a todos los recursos y asignar permisos a otros. Un colaborador tiene el mismo nivel de acceso a todos los recursos, pero no puede asignar permisos. Un lector solo puede ver los recursos de Azure existentes y un administrador de cuentas de usuario puede administrar el acceso a los recursos de Azure.

Los roles integrados más pormenorizados, como colaborador de zona DNS , tienen derechos limitados a un único servicio. Las entidades de seguridad pueden asumir cualquier número de roles.

Ámbitos

Los roles se pueden aplicar a un conjunto restringido de recursos dentro de Azure. Por ejemplo, al aplicar el ámbito al ejemplo anterior de lectura desde una cola de Service Bus, puede restringir el permiso a una sola cola: "Leer mensajes desde el punto de conexión de Azure Service Bus `blah.servicebus.windows.net/queue1`".

El ámbito puede ser tan estrecho como un único recurso o se puede aplicar a todo un grupo de recursos, una suscripción o incluso un grupo de administración.

Al probar si una entidad de seguridad tiene ciertos permisos, se tiene en cuenta la combinación de rol y ámbito. Esta combinación proporciona un mecanismo de autorización eficaz.

Negar

Anteriormente, solo se permitían reglas "permitidas" para el Control de Acceso Basado en Roles (RBAC). Este comportamiento hizo que algunos ámbitos se complicaran de construir. Por ejemplo, permitir que una entidad de seguridad acceda a todas las cuentas de almacenamiento, excepto a una, requería conceder un permiso explícito a un número potencialmente interminable de cuentas de almacenamiento. Cada vez que se creó una nueva cuenta de almacenamiento, tendría que agregarse a esta lista de cuentas. Esto agregó una sobrecarga de administración que ciertamente no era deseable.

Las reglas de denegación tienen prioridad sobre las reglas de permiso. Ahora, la representación del mismo ámbito de “permitir todo menos uno” podría describirse como dos reglas: “permitir todo” y “denegar a este específico”. Las reglas de denegación no solo facilitan la administración, sino que permiten recursos que son más seguros al denegar el acceso a todos.

Comprobación del acceso

Como puede imaginar, tener un gran número de roles y ámbitos puede hacer que averiguar los permisos efectivos de un principal de servicio sea bastante difícil. Apilar reglas de denegación sobre eso solo sirve para aumentar la complejidad. Afortunadamente, hay una calculadora de permisos que permite visualizar los permisos efectivos para cualquier principal de servicio. Normalmente se encuentra en la pestaña IAM del portal, como se muestra en la figura 9-3.

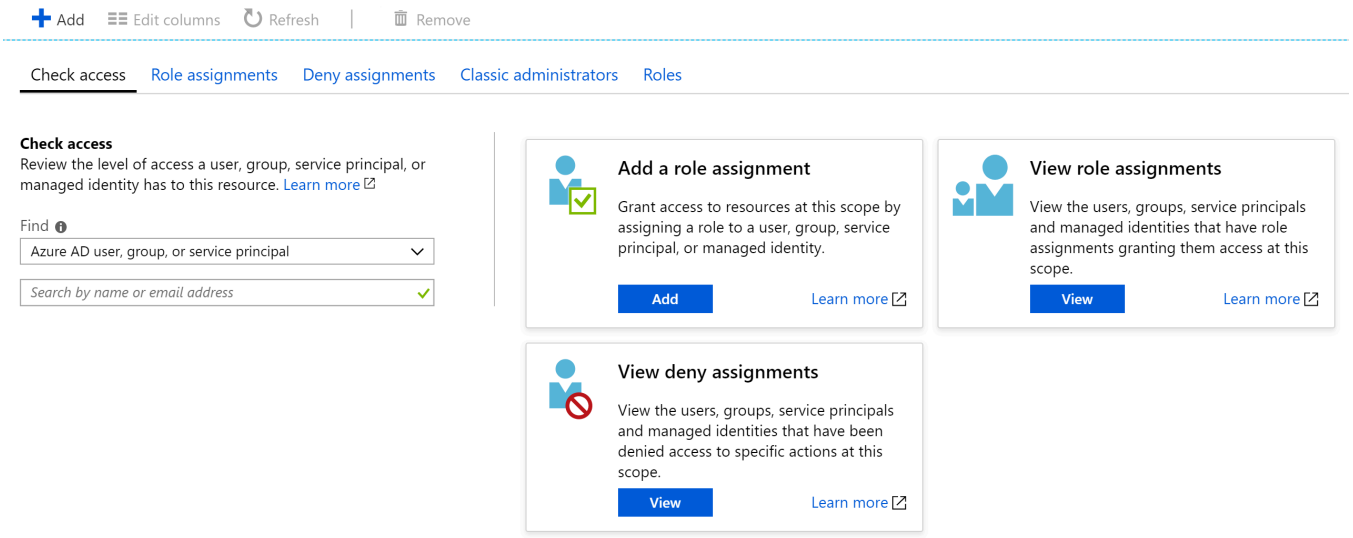


Figura 9-4. Calculadora de permisos para un servicio de aplicaciones.

Protección de secretos

Las contraseñas y los certificados son un vector de ataque común para los atacantes. El hardware de descifrado de contraseñas puede realizar un ataque por fuerza bruta e intentar adivinar miles de millones de contraseñas por segundo. Por lo tanto, es importante que las contraseñas que se usan para acceder a los recursos sean seguras, con una gran variedad de caracteres. Estas contraseñas son exactamente el tipo de contraseñas casi imposibles de recordar. Afortunadamente, las contraseñas de Azure no necesitan ser conocidas por ningún usuario.

Muchos expertos en seguridad sugieren que usar un administrador de contraseñas para mantener sus propias contraseñas es el mejor enfoque. Aunque centraliza las contraseñas en una ubicación, también permite usar contraseñas muy complejas y asegurarse de que son únicas para cada cuenta. El mismo sistema existe en Azure: un almacén central para secretos.

Azure Key Vault

Azure Key Vault proporciona una ubicación centralizada para almacenar contraseñas para cosas como bases de datos, claves de API y certificados. Una vez que se introduce un secreto en la Bóveda, nunca se muestra de nuevo, y los comandos para extraerlo y verlo son intencionadamente complicados. La información de la caja fuerte está protegida mediante el cifrado de software o los módulos de seguridad de hardware validados FIPS 140-2 Nivel 2.

El acceso a la bóveda de claves se proporciona a través de los RBACs, lo que significa que no cualquier usuario puede acceder a la información de la bóveda. Supongamos que una aplicación web desea acceder a la cadena de conexión de base de datos almacenada en Azure Key Vault. Para obtener acceso, las aplicaciones necesitan ejecutarse a través de una entidad de servicio. Bajo este rol asumido, pueden leer los secretos de la caja fuerte. Hay varias configuraciones de seguridad diferentes que pueden limitar aún más el acceso que una aplicación tiene al almacén, de modo que no pueda actualizar secretos sino que solo pueda leerlos.

Se puede supervisar el acceso a la bóveda de claves para asegurarse de que solo las aplicaciones esperadas accedan a la bóveda. Los registros se pueden integrar de nuevo en Azure Monitor, lo que permite configurar alertas cuando se encuentran condiciones inesperadas.

Kubernetes

En Kubernetes, hay un servicio similar para mantener pequeños fragmentos de información secreta. Los secretos de Kubernetes se pueden establecer a través del archivo ejecutable típico `kubectl` .

Crear un secreto es tan sencillo como buscar la versión base64 de los valores que se van a almacenar:

Consola

```
echo -n 'admin' | base64
YWRtaW4=
echo -n '1f2d1e2e67df' | base64
MWYyZDFlMmU2N2Rm
```

A continuación, agregarlo a un archivo de secretos denominado `secret.yml` que, por ejemplo, es similar al ejemplo siguiente:

yml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Por último, este archivo se puede cargar en Kubernetes mediante la ejecución del siguiente comando:

Consola

```
kubectl apply -f ./secret.yaml
```

Estos secretos se pueden integrar en volúmenes o exponerse a procesos del contenedor a través de variables de entorno. El enfoque de la aplicación Doce factores para construir aplicaciones sugiere usar el denominador común más bajo para transmitir la configuración a una aplicación. Las variables de entorno son el denominador común más bajo, ya que se admiten independientemente del sistema operativo o la aplicación.

Una alternativa a usar los secretos integrados de Kubernetes es acceder a los secretos de Azure Key Vault desde Kubernetes. La manera más sencilla de hacerlo es asignar un rol de RBAC al contenedor que busca cargar secretos. Después, la aplicación puede usar las API de Azure Key Vault para acceder a los secretos. Sin embargo, este enfoque requiere modificaciones en el código y no sigue el patrón de uso de variables de entorno. En su lugar, es posible insertar valores en un contenedor. Este enfoque es realmente más seguro que usar los secretos de Kubernetes directamente, ya que los usuarios del clúster pueden acceder a ellos.

Cifrado en tránsito y en reposo

Mantener los datos seguros es importante si está en disco o transitando entre varios servicios diferentes. La manera más eficaz de evitar que los datos se filtren es cifrarlos en un formato que otros usuarios no puedan leer fácilmente. Azure admite una amplia gama de opciones de cifrado.

En tránsito

Hay varias maneras de cifrar el tráfico en la red en Azure. Normalmente, el acceso a los servicios de Azure se realiza a través de conexiones que usan seguridad de la capa de transporte (TLS). Por ejemplo, todas las conexiones a las API de Azure requieren conexiones TLS. Del mismo modo, las conexiones a extremos en Azure Storage se pueden restringir para que funcionen solo a través de conexiones cifradas TLS.

TLS es un protocolo complicado y simplemente sabe que la conexión usa TLS no es suficiente para garantizar la seguridad. Por ejemplo, TLS 1.0 es crónicamente inseguro y TLS 1.1 no es mucho mejor. Incluso dentro de las versiones de TLS, hay varias opciones de configuración que pueden facilitar el descifrado de las conexiones. El mejor curso de acción es comprobar y ver si la conexión del servidor usa up-to-date y protocolos bien configurados.

Esta comprobación se puede realizar mediante un servicio externo, como la prueba de servidor SSL de los laboratorios SSL. Una ejecución de prueba en un punto de conexión típico de Azure, en este caso, un punto de conexión de Service Bus produce una puntuación casi perfecta de A.

Incluso los servicios como las bases de datos de Azure SQL usan el cifrado TLS para mantener ocultos los datos. La parte interesante sobre el cifrado de los datos en tránsito mediante TLS es que no es posible, incluso para Microsoft, escuchar la conexión entre ordenadores que ejecutan TLS. Esto debe proporcionar tranquilidad a las empresas preocupadas por que sus datos puedan estar en riesgo, ya sea por parte de la propia Microsoft o incluso por un actor estatal con más recursos que el atacante estándar.

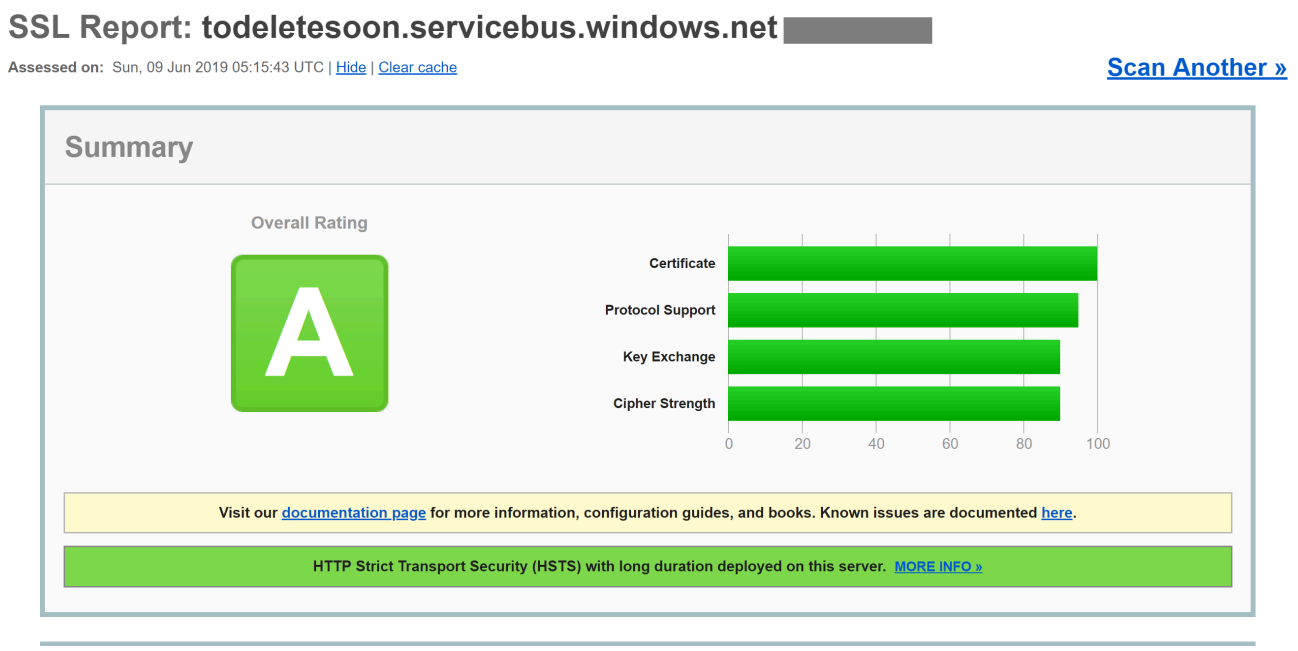


Figura 9-5. Informe de SSL Labs que muestra una calificación de A para un punto de conexión de Service Bus.

Aunque este nivel de cifrado no va a ser suficiente durante todo el tiempo, debe inspirar la confianza de que las conexiones TLS de Azure son bastante seguras. Azure seguirá evolucionando sus estándares de seguridad a medida que mejora el cifrado. Es agradable saber que hay alguien que observa los estándares de seguridad y actualiza Azure a medida que mejoran.

En reposo

En cualquier aplicación, hay una serie de lugares donde los datos se encuentran en el disco. El propio código de aplicación se carga desde algún mecanismo de almacenamiento. La mayoría de las aplicaciones también usan algún tipo de base de datos, como SQL Server, Cosmos DB o incluso el increíblemente rentable Table Storage. Todas estas bases de datos usan almacenamiento muy cifrado para asegurarse de que nadie que no sea las aplicaciones con permisos adecuados pueda leer los datos. Incluso los operadores del sistema no pueden leer datos cifrados. Por lo tanto, los clientes pueden permanecer seguros de que su información secreta permanece secreta.

Almacenamiento

La base de gran parte de Azure es el motor de Azure Storage. Los discos de máquina virtual se montan sobre Azure Storage. Azure Kubernetes Service se ejecuta en máquinas virtuales que, por sí mismas, se hospedan en Azure Storage. Incluso las tecnologías sin servidor, como Azure Functions Apps y Azure Container Instances, se ejecutan sin disco que forma parte de Azure Storage.

Si Azure Storage está bien cifrado, proporciona una base para que prácticamente todo lo demás también se cifre. Azure Storage se cifra con AES de 256 bits compatibles con FIPS 140-2. Esta es una tecnología de cifrado bien considerada que ha sido objeto de un amplio examen académico en los últimos 20 años. En la actualidad, no hay ningún ataque práctico conocido que permitiría a alguien sin conocimiento de la clave leer los datos cifrados por AES.

De forma predeterminada, Microsoft administra las claves que se usan para cifrar Azure Storage. Existen amplias protecciones para asegurarse de evitar el acceso malintencionado a estas claves. Sin embargo, los usuarios con requisitos de cifrado concretos también pueden proporcionar sus propias claves de almacenamiento que se administran en Azure Key Vault. Estas claves se pueden revocar en cualquier momento, lo que haría que el contenido de la cuenta de almacenamiento sea inaccesible al utilizar dichas claves.

Las máquinas virtuales usan almacenamiento cifrado, pero es posible proporcionar otra capa de cifrado mediante tecnologías como BitLocker en Windows o DM-Crypt en Linux. Estas tecnologías significan que incluso si la imagen de disco se ha filtrado del almacenamiento, sería casi imposible leerla.

Azure SQL

Las bases de datos hospedadas en Azure SQL usan una tecnología denominada Cifrado de datos transparente (TDE) para asegurarse de que los datos permanecen cifrados. Está habilitado de forma predeterminada en todas las bases de datos SQL recién creadas, pero debe habilitarse manualmente para las bases de datos heredadas. TDE ejecuta el cifrado en tiempo real y el descifrado de no solo la base de datos, sino también las copias de seguridad y los registros de transacciones.

Los parámetros de cifrado se almacenan en la `master` base de datos y, al iniciarse, se leen en memoria para las operaciones restantes. Esto significa que la `master` base de datos debe permanecer sin cifrar. Microsoft administra la clave real. Sin embargo, los usuarios con requisitos de seguridad exactos pueden proporcionar su propia clave en Key Vault de la misma manera que para Azure Storage. Key Vault proporciona servicios como rotación y revocación de claves.

La parte "Transparente" de TDS procede del hecho de que no hay cambios de cliente necesarios para usar una base de datos cifrada. Aunque este enfoque proporciona una buena seguridad, la pérdida de la contraseña de la base de datos es suficiente para que los usuarios puedan descifrar los datos. Hay otro enfoque que cifra columnas o tablas individuales en una base de datos. Always Encrypted garantiza que, en ningún momento, los datos cifrados aparezcan en texto sin formato dentro de la base de datos.

La configuración de este nivel de cifrado requiere ejecutarse mediante un asistente en SQL Server Management Studio para seleccionar el tipo de cifrado y dónde en Key Vault almacenar las claves asociadas.

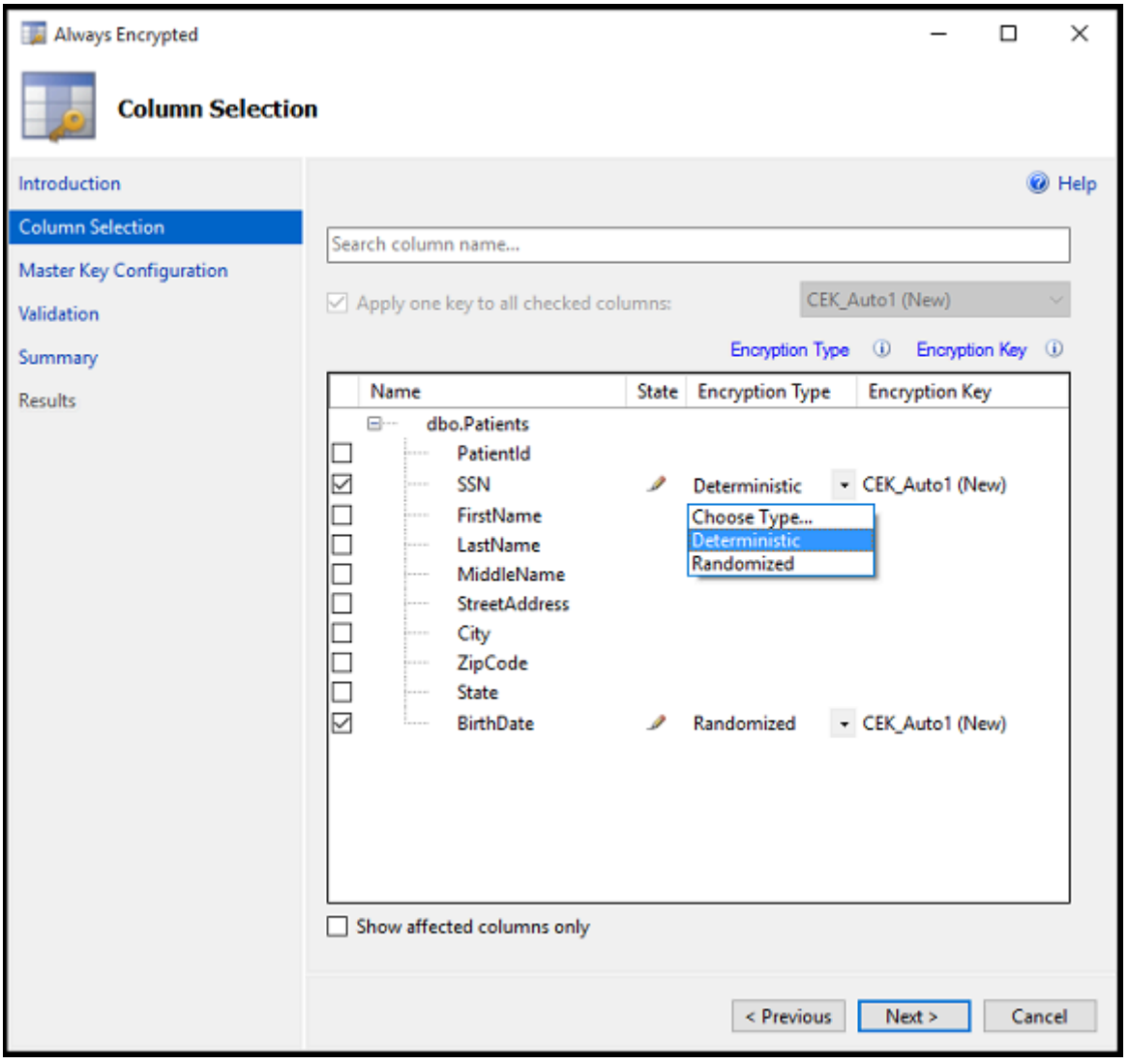


Figura 9-6. Selección de columnas en una tabla que se va a cifrar mediante Always Encrypted.

Las aplicaciones cliente que leen información de estas columnas cifradas deben realizar asignaciones especiales para leer datos cifrados. Las cadenas de conexión deben actualizarse con `Column Encryption Setting=Enabled` y las credenciales de cliente deben recuperarse de Key Vault. Después, el cliente de SQL Server debe estar preparado con las claves de cifrado de las columnas. Una vez hecho esto, las acciones restantes usan las interfaces estándar para el cliente SQL. Es decir, las herramientas como Dapper y Entity Framework, que se basan en sql Client, seguirán funcionando sin cambios. Always Encrypted puede que aún no esté disponible para todos los controladores de SQL Server en todos los lenguajes.

La combinación de TDE y Always Encrypted, ambas que se pueden usar con claves específicas del cliente, garantizan que se admita incluso los requisitos de cifrado más exactos.

Cosmos DB

Cosmos DB es la base de datos más reciente proporcionada por Microsoft en Azure. Se ha construido desde cero con seguridad y criptografía en mente. El cifrado AES-256bit es estándar para todas las bases de datos de Cosmos DB y no se puede deshabilitar. Junto con el requisito tls 1.2 para la comunicación, se cifra toda la solución de almacenamiento.

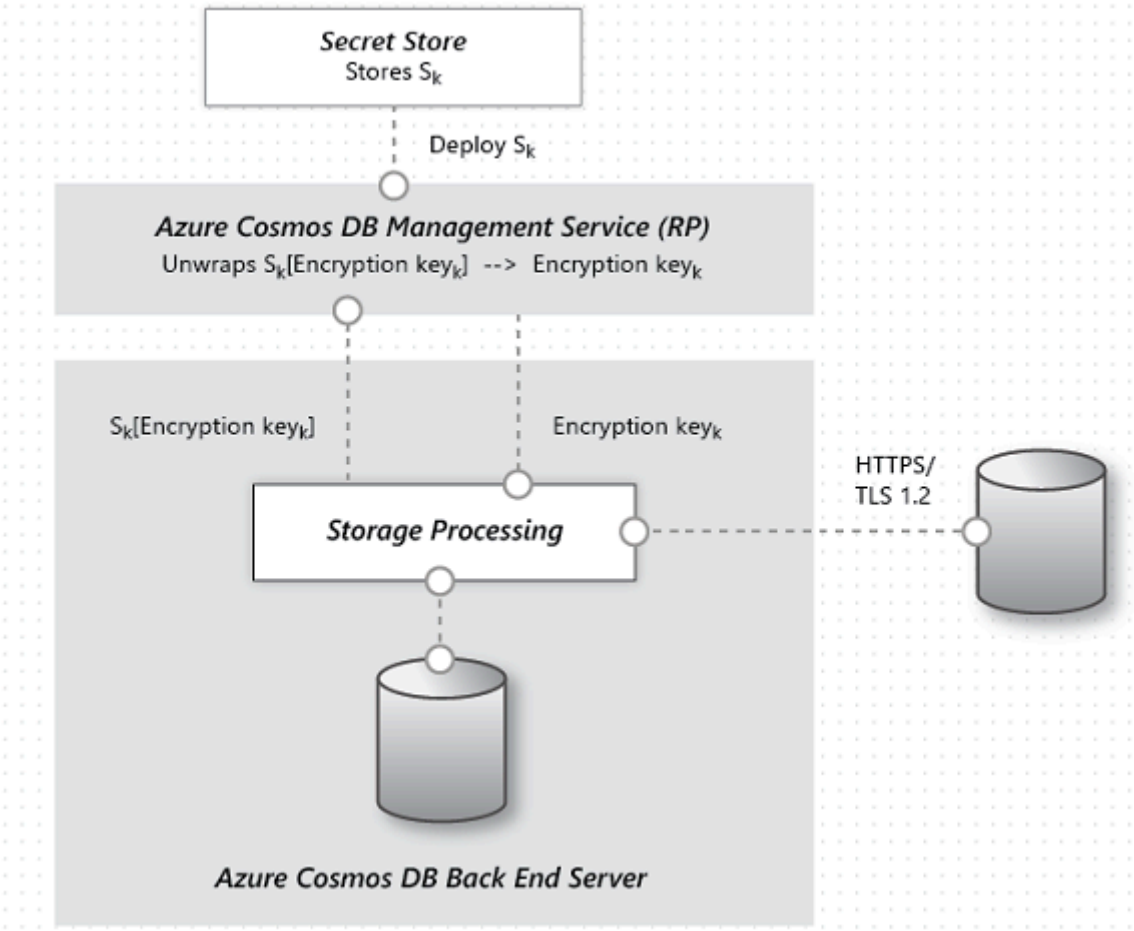


Figura 9-7. Flujo de cifrado de datos en Cosmos DB.

Aunque Cosmos DB no proporciona claves de cifrado de cliente, el equipo ha realizado un trabajo significativo para asegurarse de que sigue siendo PCI-DSS compatible sin eso. Cosmos DB tampoco admite ningún tipo de cifrado de columna única similar a Always Encrypted de Azure SQL todavía.

Mantener la seguridad

Azure tiene todas las herramientas necesarias para liberar un producto altamente seguro. Sin embargo, una cadena solo es tan fuerte como su vínculo más débil. Si las aplicaciones implementadas encima de Azure no se desarrollan con una mentalidad de seguridad adecuada y auditorías de seguridad adecuadas, se convierten en el vínculo débil de la cadena. Hay muchas herramientas de análisis estáticos excelentes, bibliotecas de cifrado y prácticas de seguridad que se pueden usar para garantizar que el software instalado en Azure sea tan seguro como Azure. Algunos ejemplos son las herramientas de análisis estático y las bibliotecas de cifrado.

Siguiente anterior

DevOps

17/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

El lema favorito de los consultores de software es responder a "Depende" de cualquier pregunta planteada. No se debe a que los consultores de software están encantados de no tomar una posición. Es porque no hay ninguna respuesta verdadera a ninguna pregunta en el software. No hay ningún derecho absoluto y incorrecto, sino un equilibrio entre los opuestos.

Por ejemplo, las dos principales escuelas de desarrollo de aplicaciones web: Aplicaciones de página única (SPA) frente a aplicaciones del lado servidor. Por un lado, la experiencia del usuario tiende a ser mejor con las SPA y la cantidad de tráfico al servidor web se puede minimizar, lo que permite hospedarlos en algo tan sencillo como el hospedaje estático. Por otro lado, las SPA tienden a ser más lentas para desarrollarse y ser más difíciles de probar. ¿Cuál es la opción correcta? Depende de tu situación.

Las aplicaciones nativas de la nube no son inmunes a esa misma dictomía. Tienen ventajas claras en términos de velocidad de desarrollo, estabilidad y escalabilidad, pero administrarlas puede ser bastante más difícil.

Hace años, no era raro que el proceso de mover una aplicación de desarrollo a producción tarde un mes o incluso más. Las empresas lanzaron software durante un período de 6 meses o incluso cada año. No hay más que fijarse en Microsoft Windows para hacerse una idea de la cadencia de las versiones que eran aceptables antes de que llegara la estabilidad de Windows 10. Cinco años pasaron entre Windows XP y Vista, tres más entre Vista y Windows 7.

Ahora está bastante bien establecido que ser capaz de lanzar software rápidamente ofrece a las empresas de movimiento rápido una gran ventaja de mercado sobre sus competidores más ranurados. Por ese motivo, las actualizaciones principales de Windows 10 ahora son aproximadamente cada seis meses.

Los patrones y prácticas que permiten versiones más rápidas y confiables para ofrecer valor a la empresa se conocen colectivamente como DevOps. Constan de una amplia gama de ideas que abarcan todo el ciclo de vida de desarrollo de software, desde la especificación de una aplicación hasta la entrega y el funcionamiento de esa aplicación.

DevOps surgió antes de los microservicios y es probable que el movimiento hacia servicios más pequeños y más adecuados para propósitos no hubiera sido posible sin DevOps para facilitar la publicación y el funcionamiento no solo de una sino muchas aplicaciones en producción.

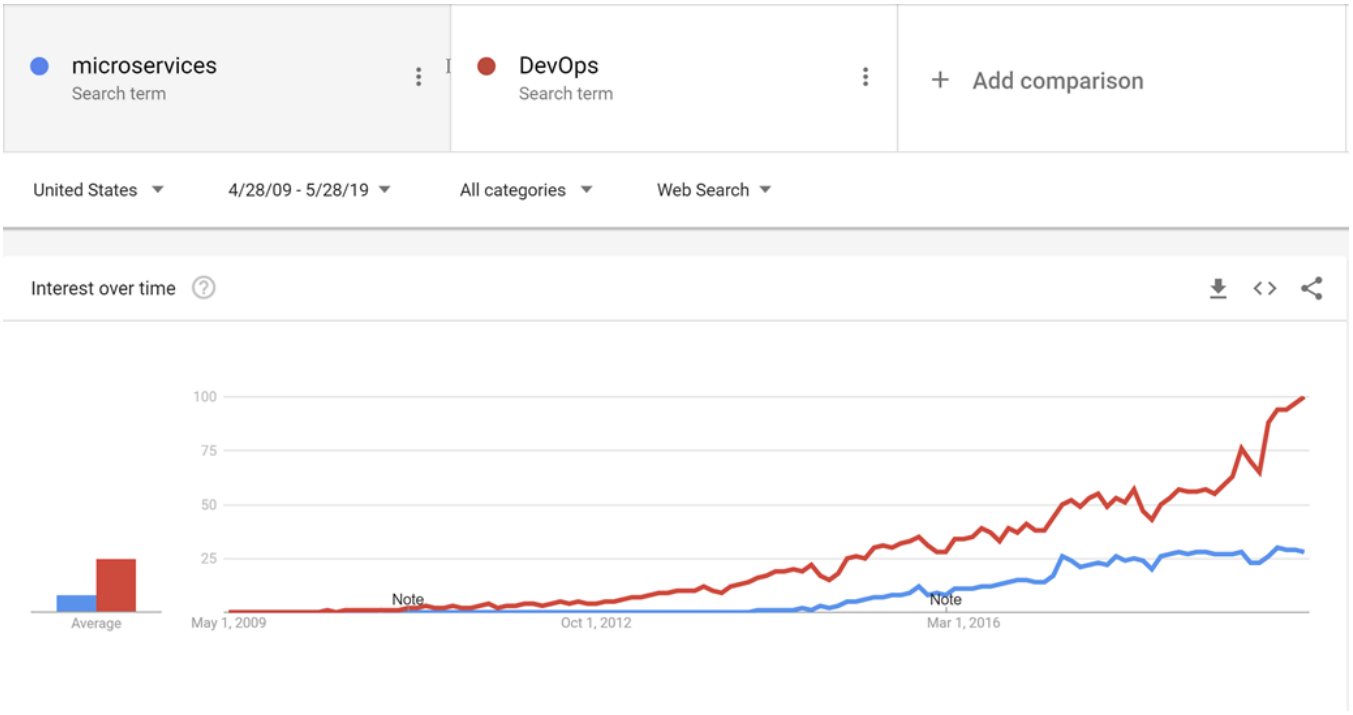


Figura 10-1 : DevOps y microservicios.

Gracias a unos procedimientos adecuados de DevOps, se puede disfrutar de las ventajas de las aplicaciones nativas de nube sin verse inundado por un aluvión de trabajo para poner en marcha las aplicaciones.

No hay ningún martillo dorado cuando se trata de DevOps. Nadie puede vender una solución completa y integral para lanzar y operar aplicaciones de alta calidad. Esto se debe a que cada aplicación es muy diferente de todas las demás. Sin embargo, hay herramientas que pueden hacer que DevOps sea una propuesta mucho menos intimidante. Una de estas herramientas se conoce como Azure DevOps.

Azure DevOps

Azure DevOps tiene un pedigrí largo. Sus inicios se remontan a cuando Team Foundation Server pasó al modo en línea por primera vez, y se ha sometido a distintos cambios de nombre: Visual Studio Online y Visual Studio Team Services. Con los años, sin embargo, se ha convertido en algo mucho más que sus predecesores.

Azure DevOps se divide en cinco componentes principales:

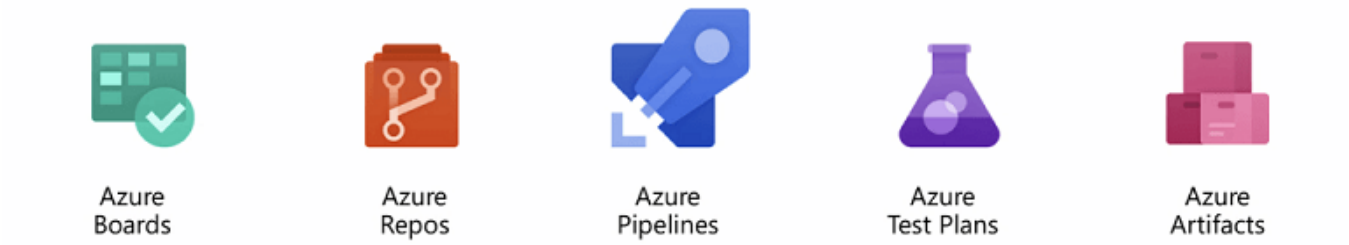


Figura 10-2 : Azure DevOps.

Azure Repos — administración de código fuente que admite el veterano Team Foundation Version Control (TFVC) y el favorito del sector, Git. Las solicitudes de incorporación de cambios proporcionan una manera de habilitar la codificación social fomentando la discusión de los cambios a medida que se realizan.

Azure Boards : proporciona una herramienta de seguimiento de problemas y elementos de trabajo que se esfuerza por permitir a los usuarios elegir los flujos de trabajo que funcionan mejor para ellos. Viene con una serie de plantillas preconfiguradas, incluidas las que admiten los estilos SCRUM y Kanban de desarrollo.

Azure Pipelines : un sistema de administración de versiones y compilación que admite una estrecha integración con Azure. Las compilaciones se pueden ejecutar en varias plataformas de Windows a Linux a macOS. Los agentes de compilación se pueden aprovisionar en la nube o en el entorno local.

Planes de pruebas de Azure. No se dejará a ninguna persona de Control de Calidad atrás con la gestión de pruebas y el soporte para pruebas exploratorias que ofrece la función de Planes de prueba.

Azure Artifacts - un feed de artefactos que permite a las empresas crear sus propias versiones internas de NuGet, npm y otras. Sirve para el doble propósito de actuar como una memoria caché de paquetes superiores si se produce un fallo en un repositorio centralizado.

La unidad organizativa de nivel superior de Azure DevOps se conoce como Project. Dentro de cada proyecto, los distintos componentes, como Azure Artifacts, se pueden activar y desactivar. Cada uno de estos componentes proporciona diferentes ventajas para las aplicaciones nativas de la nube. Los tres más útiles son repositorios, paneles y canalizaciones. Si los usuarios quieren administrar su código fuente en otra pila de repositorios, como GitHub, pero siguen aprovechando Azure Pipelines y otros componentes, eso es perfectamente posible.

Afortunadamente, los equipos de desarrollo tienen muchas opciones al seleccionar un repositorio. Uno de ellos es GitHub.

Acciones de GitHub

Fundada en 2009, GitHub es un repositorio ampliamente popular basado en web para hospedar proyectos, documentación y código. Muchas grandes empresas tecnológicas, como Apple, Amazon, Google y las corporaciones estándar usan GitHub. GitHub usa el sistema de control de versiones distribuido de código abierto denominado Git como base. Además, agrega su propio conjunto de características, como seguimiento de defectos, solicitudes de incorporación de cambios, administración de tareas y wikis para cada base de código.

A medida que GitHub evoluciona, también agrega características de DevOps. Por ejemplo, GitHub tiene su propia canalización de integración continua/entrega continua (CI/CD), denominada `GitHub Actions`. Acciones de GitHub es una herramienta de automatización de flujos de trabajo impulsada por la comunidad. Permite que los equipos de DevOps se integren con sus herramientas existentes, mezclen y combinen nuevos productos, y se integren con su ciclo de vida de software, incluidos los socios de CI/CD existentes.

GitHub tiene más de 40 millones de usuarios, lo que lo convierte en el host más grande del código fuente del mundo. En octubre de 2018, Microsoft compró GitHub. Microsoft ha prometido que GitHub seguirá siendo una plataforma abierta que cualquier desarrollador pueda conectar y ampliar. Sigue funcionando como una empresa independiente. GitHub ofrece planes para cuentas empresariales, de equipo, profesionales y gratuitas.

Control de código fuente

Organizar el código para una aplicación nativa de la nube puede ser difícil. En lugar de una sola aplicación gigante, las aplicaciones nativas de la nube tienden a estar formadas por una web de aplicaciones más pequeñas que se comunican entre sí. Al igual que con todas las cosas de la computación, la mejor disposición del código sigue siendo una pregunta abierta. Hay ejemplos de aplicaciones correctas que usan diferentes tipos de diseños, pero dos variantes parecen tener la mayor popularidad.

Antes de entrar en el propio control de código fuente real, es probable que valga la pena decidir cuántos proyectos son adecuados. Dentro de un mismo proyecto existe compatibilidad con varios repositorios y canalizaciones de compilación. Los paneles son un poco más complicados, pero en ellos las tareas también se pueden asignar fácilmente a varios equipos dentro de un mismo proyecto. Es posible admitir a cientos o incluso miles de desarrolladores desde un único proyecto de Azure DevOps. Es probable que sea el mejor enfoque, ya que proporciona un único lugar para que todos los desarrolladores trabajen y reduce la confusión

al intentar encontrar esa aplicación cuando los desarrolladores no están seguros de en qué proyecto reside.

La división de código para microservicios dentro del proyecto de Azure DevOps puede ser un poco más difícil.

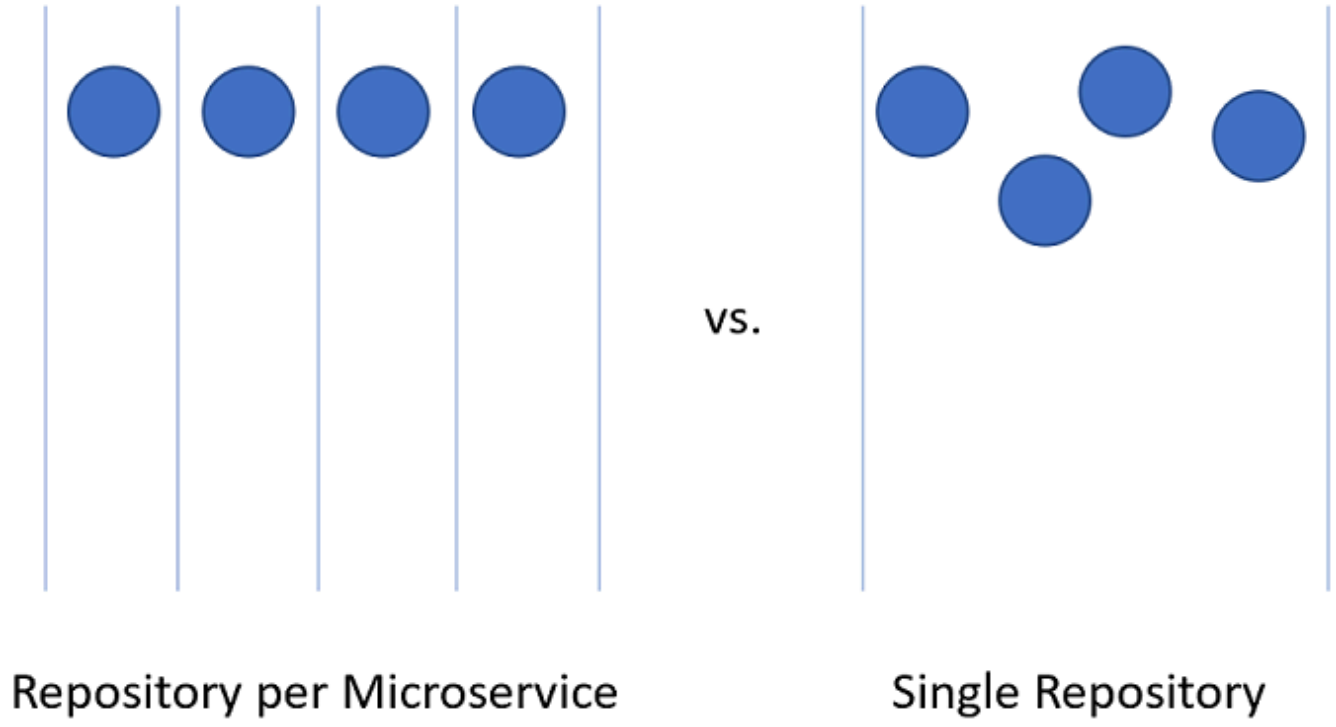


Figura 10-3 : uno frente a muchos repositorios.

Repositorio para cada microservicio

A primera vista, este enfoque parece el enfoque más lógico para dividir el código fuente de los microservicios. Cada repositorio puede contener el código necesario para compilar el microservicio. Las ventajas de este enfoque son fácilmente visibles:

1. Las instrucciones para compilar y mantener la aplicación se pueden agregar a un archivo LÉAME en la raíz de cada repositorio. Al desplazarse por los repositorios, es fácil encontrar estas instrucciones, lo que reduce el tiempo de puesta en marcha para los desarrolladores.
2. Cada servicio se encuentra en un lugar lógico, fácilmente encontrado al conocer el nombre del servicio.
3. Las compilaciones se pueden configurar fácilmente de modo que solo se desencadenen cuando se realiza un cambio en el repositorio propietario.
4. El número de cambios que entran en un repositorio se limita al pequeño número de desarrolladores que trabajan en el proyecto.
5. La seguridad es fácil de configurar mediante la restricción de los repositorios a los que los desarrolladores tienen permisos de lectura y escritura.

6. El equipo propietario puede cambiar la configuración del nivel de repositorio con un mínimo de discusión con otros usuarios.

Una de las ideas clave detrás de los microservicios es que los servicios deben estar siloados y separados entre sí. Al usar el diseño controlado por dominio para decidir los límites de los servicios, los servicios actúan como límites transaccionales. Las actualizaciones de base de datos no deben abarcar varios servicios. Esta colección de datos relacionados se conoce como contexto delimitado. Esta idea se refleja mediante el aislamiento de los datos de microservicios en una base de datos independiente y autónoma del resto de los servicios. Tiene mucho sentido llevar esta idea hasta el código fuente.

Sin embargo, este enfoque no es sin sus problemas. Uno de los problemas de desarrollo más complicados de nuestra época es administrar las dependencias. Tenga en cuenta el número de archivos que componen el directorio promedio `node_modules` . Una nueva instalación de algo como `create-react-app` es probable que traiga con él miles de paquetes. La cuestión de cómo administrar estas dependencias es difícil.

Si se actualiza una dependencia, los paquetes de bajada también deben actualizar esta dependencia. Desafortunadamente, eso toma el trabajo de desarrollo, por lo que, invariablemente, el `node_modules` directorio termina con varias versiones de un único paquete, cada uno de ellos una dependencia de algún otro paquete que se versiona con una cadencia ligeramente diferente. Al implementar una aplicación, ¿qué versión de una dependencia se debe usar? ¿La versión que está actualmente en producción? ¿La versión que está actualmente en versión beta, pero que probablemente esté en producción en el momento en que el consumidor la lleve a producción? Problemas difíciles que no se resuelven mediante el uso de microservicios.

Hay bibliotecas que dependen de una amplia variedad de proyectos. Al dividir los microservicios con uno en cada repositorio, las dependencias internas se pueden resolver mejor mediante el repositorio interno, Azure Artifacts. Las compilaciones para bibliotecas insertarán sus versiones más recientes en Azure Artifacts para su consumo interno. El proyecto descendente se debe seguir actualizando manualmente para establecer una dependencia con los paquetes recién actualizados.

Otra desventaja se presenta al mover código entre servicios. Aunque sería genial creer que la primera vez que una aplicación se divide en microservicios todo sale a pedir de boca, la realidad es que rara vez somos tan clarividentes como para no cometer errores al dividir servicios. Por lo tanto, la funcionalidad y el código que la impulsa tendrán que pasar de servicio a servicio y de repositorio a repositorio. Al pasar de un repositorio a otro, el código pierde su historial. Hay muchos casos, especialmente en el caso de una auditoría, donde tener un historial completo en un fragmento de código es inestimable.

La desventaja final y más importante es coordinar los cambios. En una aplicación verdadera de microservicios, no debe haber dependencias de implementación entre los servicios. Debe ser posible implementar los servicios A, B y C en cualquier orden, ya que tienen acoplamiento flexible. Sin embargo, en realidad, hay ocasiones en las que es deseable realizar un cambio que cruce varios repositorios al mismo tiempo. Algunos ejemplos incluyen actualizar una biblioteca para cerrar un agujero de seguridad o cambiar un protocolo de comunicación usado por todos los servicios.

Para realizar un cambio entre repositorios, es necesario realizar una confirmación en cada repositorio sucesivamente. Cada cambio en cada repositorio deberá solicitarse y revisarse por separado. Esta actividad puede ser difícil de coordinar.

Una alternativa al uso de muchos repositorios es colocar todo el código fuente junto en un único repositorio gigante y omnisciente.

Repositorio único

En este enfoque, a veces denominado monorepository, todo el código fuente de cada servicio se coloca en el mismo repositorio. Al principio, este enfoque parece una idea terrible que probablemente haga que manejar el código fuente sea complicado. Sin embargo, hay algunas ventajas marcadas para trabajar de esta manera.

La primera ventaja es que es más fácil administrar las dependencias entre proyectos. En lugar de confiar en alguna fuente de artefactos externos, los proyectos pueden importarse directamente entre sí. Esto significa que las actualizaciones son instantáneas y es probable que las versiones en conflicto se encuentren en tiempo de compilación en la estación de trabajo del desarrollador. Dicho de otro modo: parte de las pruebas de integración se realizarían en las primeras fases de desarrollo.

Al mover código entre proyectos, ahora es más fácil conservar el historial, ya que los archivos se detectarán como movidos en lugar de volver a escribirse.

Otra ventaja es que se pueden realizar cambios de gran alcance que cruzan los límites del servicio en un solo compromiso. Esta actividad reduce la sobrecarga de tener potencialmente docenas de cambios para revisar individualmente.

Hay muchas herramientas que pueden realizar análisis estáticos del código para detectar prácticas de programación no seguras o un uso problemático de las API. En un mundo de varios repositorios, cada repositorio tendrá que iterarse para encontrar los problemas en ellos. El único repositorio permite ejecutar el análisis en un solo lugar.

También hay muchas desventajas en el enfoque de repositorio único. Uno de los más preocupantes es que tener un único repositorio genera problemas de seguridad. Si el

contenido de un repositorio se filtra en un modelo de repositorio por servicio, la cantidad de código perdido es mínima. Con un único repositorio, todo lo que posee la empresa podría perderse. Ha habido muchos ejemplos en el pasado de este tipo de situaciones que han descarrilado los esfuerzos completos de desarrollo de juegos. Tener varios repositorios expone menos área expuesta, que es un rasgo deseable en la mayoría de las prácticas de seguridad.

Es probable que el tamaño del repositorio único se vuelva inadministrable rápidamente. Esto presenta algunas implicaciones interesantes sobre el rendimiento. Es posible que sea necesario usar herramientas especializadas como El sistema de archivos virtuales para Git, que originalmente se diseñó para mejorar la experiencia de los desarrolladores en el equipo de Windows.

Con frecuencia, el argumento para usar un único repositorio se reduce a un argumento que Facebook o Google usan este método para la organización del código fuente. Si el enfoque es lo suficientemente bueno para estas empresas, seguramente, es el enfoque correcto para todas las empresas. La verdad del asunto es que pocas empresas operan en cualquier cosa como la escala de Facebook o Google. Los problemas que se producen a esas escalas son diferentes a los que se enfrentan la mayoría de los desarrolladores. Lo que es bueno para el ganso puede no ser bueno para el gander.

Al final, cualquiera de las soluciones se puede usar para hospedar el código fuente de los microservicios. Sin embargo, en la mayoría de los casos, la administración y la sobrecarga de ingeniería del funcionamiento en un único repositorio no merecen la pena las ventajas mínimas. Dividir el código en varios repositorios fomenta una mejor separación de preocupaciones y fomenta la autonomía entre los equipos de desarrollo.

Estructura de directorios estándar

Independientemente del debate único frente a varios repositorios, cada servicio tendrá su propio directorio. Una de las mejores optimizaciones para permitir que los desarrolladores crucen entre proyectos rápidamente es mantener una estructura de directorios estándar.

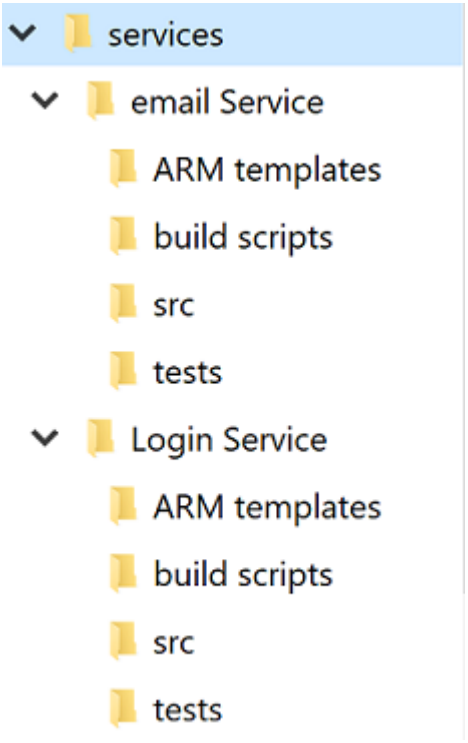


Figura 10-4 : estructura de directorio estándar.

Cada vez que se crea un nuevo proyecto, se debe usar una plantilla que coloca la estructura correcta. Esta plantilla también puede incluir elementos útiles como un archivo README modelo y un `azure-pipelines.yml`. En cualquier arquitectura de microservicios, un alto grado de varianza entre proyectos dificulta las operaciones masivas con los servicios.

Hay muchas herramientas que pueden proporcionar plantillas para un directorio completo, que contiene varios directorios de código fuente. Yeoman es popular en el mundo de JavaScript y GitHub han publicado recientemente plantillas de repositorio, que proporcionan gran parte de la misma funcionalidad.

Administración de tareas

La administración de tareas en cualquier proyecto puede ser difícil. Por adelantado, hay innumerables preguntas que se deben responder sobre el tipo de flujos de trabajo que se van a configurar para garantizar una productividad óptima para desarrolladores.

Las aplicaciones nativas de la nube tienden a ser más pequeñas que los productos de software tradicionales o al menos se dividen en servicios más pequeños. El seguimiento de problemas o tareas relacionados con estos servicios sigue siendo tan importante como con cualquier otro proyecto de software. Nadie quiere perder el seguimiento de algún elemento de trabajo o explicar a un cliente que su problema no se registró correctamente. Los paneles se configuran en el nivel de proyecto, pero dentro de cada proyecto, se pueden definir áreas. Esto permite desglosar los problemas en varios componentes. La ventaja de mantener todo el trabajo de

toda la aplicación en un solo lugar es que es fácil mover elementos de trabajo de un equipo a otro a medida que se entienden mejor.

Azure DevOps incluye una serie de plantillas populares preconfiguradas. En la configuración más básica, todo lo que se necesita para saber es lo que hay en el trabajo pendiente, qué personas están trabajando y lo que se hace. Es importante tener esta visibilidad sobre el proceso de creación de software, de modo que el trabajo se pueda priorizar y completar las tareas notificadas al cliente. Por supuesto, algunos proyectos de software se adhieren a un proceso tan sencillo como `to do`, `doing` y `done`. No tarda mucho tiempo en empezar a agregar pasos como `QA` o `Detailed Specification` al proceso.

Una de las partes más importantes de las metodologías ágiles es la introspección automática a intervalos regulares. Estas revisiones están diseñadas para proporcionar información sobre los problemas a los que se enfrenta el equipo y cómo se pueden mejorar. Con frecuencia, esto significa cambiar el flujo de problemas y características a través del proceso de desarrollo. Por lo tanto, es perfectamente lícito ampliar los diseños de los paneles con más etapas.

Las etapas en los paneles no son la única herramienta organizativa. En función de la configuración del panel, hay una jerarquía de elementos de trabajo. El elemento más granular que puede aparecer en un panel es una tarea. De manera predeterminada, una tarea contiene campos para un título, una descripción, una prioridad, una estimación de la cantidad de trabajo restante y la capacidad para vincular otros elementos de trabajo o elementos de desarrollo (ramas, confirmaciones, solicitudes de incorporación de cambios, compilaciones, etc.). Los elementos de trabajo se pueden clasificar en diferentes áreas de la aplicación y diferentes iteraciones (sprints) para facilitar su búsqueda.

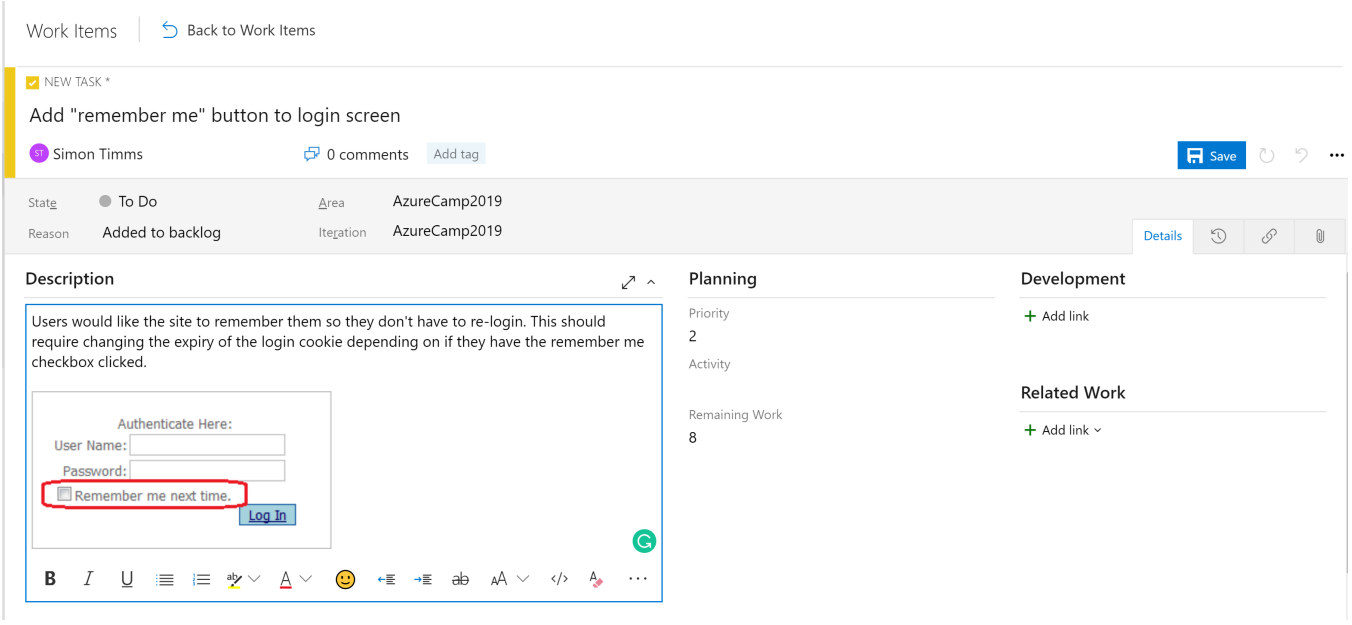


Figura 10-5 : Tarea en Azure DevOps.

El campo de descripción admite los estilos normales que se podría esperar (negrita, cursiva, subrayado y tachado) y la capacidad de insertar imágenes. Esto hace que sea una herramienta eficaz para su uso al especificar el trabajo o los errores.

Las tareas se pueden agrupar en funcionalidades, que definen una unidad de trabajo más amplia. Las características, a su vez, se pueden agrupar en epopeyas. La clasificación de tareas en esta jerarquía facilita mucho la comprensión de la proximidad de una característica grande a la implementación.

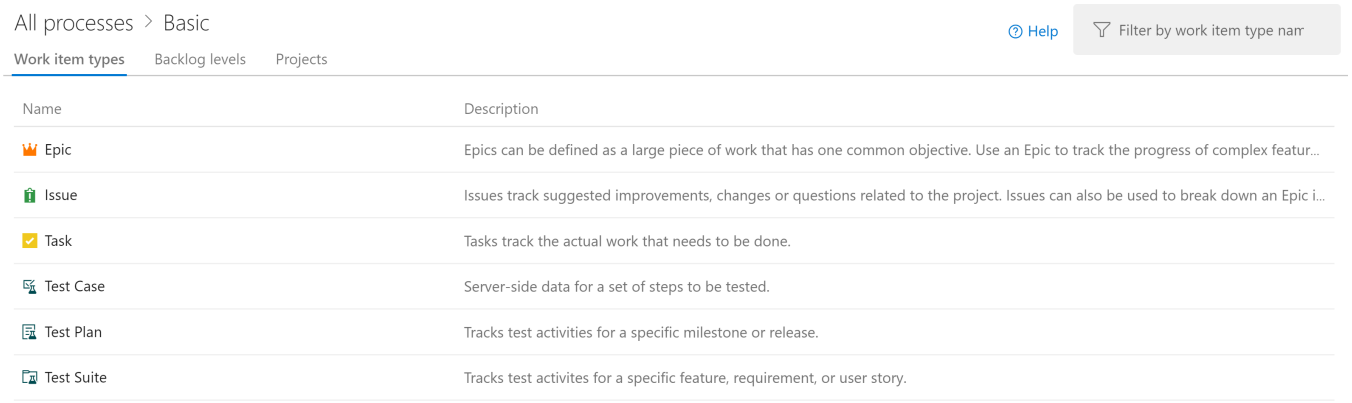


Figura 10-6 : Elemento de trabajo en Azure DevOps.

En Azure Boards existen diferentes tipos de vistas de los problemas. Los elementos que aún no están programados aparecen en el trabajo pendiente. Desde ahí, se pueden asignar a un sprint. Un sprint es un cuadro de tiempo durante el que se espera que se complete cierta cantidad de trabajo. Este trabajo puede incluir tareas, pero también la resolución de incidencias. Un sprint completo se puede administrar desde la sección Sprint del panel. Esta vista refleja cómo va avanzando el trabajo, e incluye un gráfico de evolución que proporciona una estimación en constante actualización de si el sprint se va a realizar correctamente.

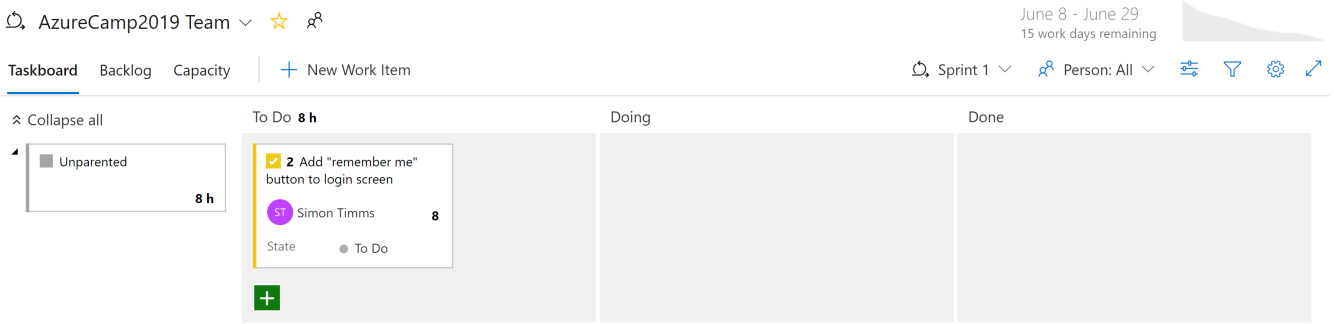


Figura 10-7 : Panel en Azure DevOps.

Por ahora, debería ser evidente que hay una gran potencia en los Paneles en Azure DevOps. Para los desarrolladores, hay vistas sencillas de lo que se está trabajando. Para los administradores de proyectos: vistas del trabajo próximo y una visión general del trabajo

actual. Para los administradores, hay muchos informes sobre recursos y capacidad. Desafortunadamente, no hay nada mágico sobre las aplicaciones nativas de la nube que eliminan la necesidad de realizar un seguimiento del trabajo. Pero si debe realizar un seguimiento del trabajo, hay algunos lugares en los que la experiencia es mejor que en Azure DevOps.

Canalizaciones de CI/CD

Casi ningún cambio en el ciclo de vida de desarrollo de software ha sido tan revolucionario como la llegada de la integración continua (CI) y la entrega continua (CD). La compilación y ejecución de pruebas automatizadas en el código fuente de un proyecto tan pronto como se comprueba un cambio detecta errores al principio. Antes de la llegada de compilaciones de integración continua, no sería raro extraer código del repositorio y encontrar que no superó pruebas ni siquiera se pudo compilar. Esto dio lugar a que se llevara un seguimiento del origen de la rotura de código.

Tradicionalmente, el envío de software al entorno de producción requería una amplia documentación y una lista de pasos. Cada uno de estos pasos debe completarse manualmente en un proceso muy propenso a errores.

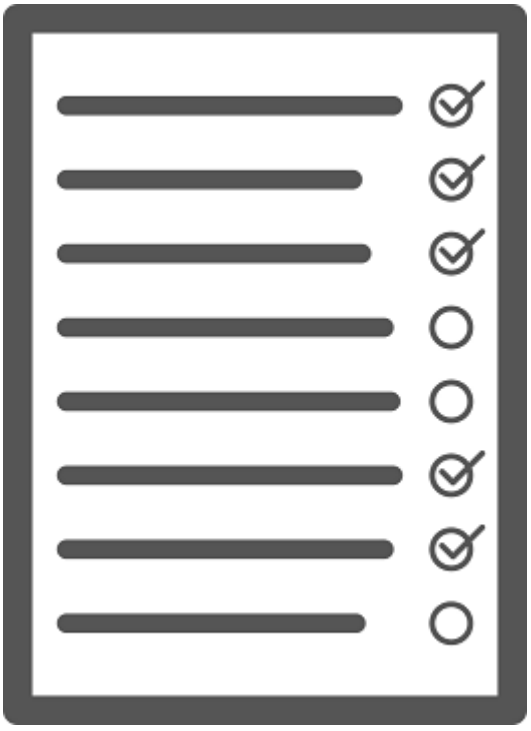


Figura 10-8 : Lista de comprobación.

La hermana de la integración continua es la entrega continua en la que los paquetes recién creados se implementan en un entorno. El proceso manual no se puede escalar para que coincida con la velocidad de desarrollo, por lo que la automatización es más importante. Las

listas de comprobación se reemplazan por scripts que pueden ejecutar las mismas tareas de forma más rápida y precisa que cualquier persona.

El entorno al que la entrega continua se realiza podría ser un entorno de prueba o, como hacen muchas empresas tecnológicas importantes, podría ser el entorno de producción. Esto último requiere una inversión en pruebas de alta calidad que pueden dar confianza de que un cambio no interrumpirá la producción para los usuarios. De la misma manera que la integración continua detectaba problemas en el código, una entrega continua temprana detecta con prontitud problemas en el proceso de implementación.

La importancia de automatizar el proceso de compilación y entrega está acentuada por las aplicaciones nativas de la nube. Las implementaciones se producen con más frecuencia y en más entornos, por lo que la implementación manual se vuelve casi imposible.

Compilaciones de Azure

Azure DevOps proporciona un conjunto de herramientas para facilitar la integración continua y la implementación más que nunca. Estas herramientas se encuentran en Azure Pipelines. La primera de ellas es Azure Builds, que es una herramienta para ejecutar definiciones de compilación basadas en YAML a escala. Los usuarios pueden traer sus propias máquinas de compilación (ideales para si la compilación requiere un entorno de configuración minuciosa) o usar una máquina desde un grupo actualizado constantemente de máquinas virtuales hospedadas en Azure. Estos agentes de compilación hospedados vienen preinstalados con una amplia gama de herramientas de desarrollo para no solo desarrollo de .NET, sino para todo, desde Java a Python hasta el desarrollo de iPhone.

DevOps incluye una amplia gama de definiciones de compilación integradas que se pueden personalizar para cualquier compilación. Las definiciones de compilación se definen en un archivo denominado `azure-pipelines.yml` y se comprueban en el repositorio para que se puedan versionarse junto con el código fuente. Esto hace que resulte mucho más fácil realizar cambios en la canalización de compilación en una rama, ya que los cambios se pueden comprobar solo en esa rama. En la figura 10-9 se muestra un ejemplo `azure-pipelines.yml` para compilar una aplicación web de ASP.NET en el marco completo.

yml

```yml
name: $(rev:r)

variables:
  version: 9.2.0.$(Build.BuildNumber)
  solution: Portals.sln
  artifactName: drop
  buildPlatform: any cpu
  buildConfiguration: release
```

```
pool:
  name: Hosted VisualStudio
  demands:
  - msbuild
  - visualstudio
  - vstest

steps:
- task: NuGetToolInstaller@0
  displayName: 'Use NuGet 4.4.1'
  inputs:
    versionSpec: 4.4.1

- task: NuGetCommand@2
  displayName: 'NuGet restore'
  inputs:
    restoreSolution: '$(solution)'

- task: VSBuild@1
  displayName: 'Build solution'
  inputs:
    solution: '$(solution)'
    msbuildArgs: '-p:DeployOnBuild=true -p:WebPublishMethod=Package -
p:PackageAsSingleFile=true -p:SkipInvalidConfigurations=true -
p:PackageLocation="$(build.artifactstagingdirectory)\\"'
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'

- task: VSTest@2
  displayName: 'Test Assemblies'
  inputs:
    testAssemblyVer2: |
     **\$(buildConfiguration)\**\*test*.dll
     !**\obj\**
     !**\*testadapter.dll
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'

- task: CopyFiles@2
  displayName: 'Copy UI Test Files to: $(build.artifactstagingdirectory)'
  inputs:
    SourceFolder: UITests
    TargetFolder: '$(build.artifactstagingdirectory)/uitests'

- task: PublishBuildArtifacts@1
  displayName: 'Publish Artifact'
  inputs:
    PathtoPublish: '$(build.artifactstagingdirectory)'
    ArtifactName: '$(artifactName)'
  condition: succeededOrFailed()
```

Figura 10-9: Ejemplo de azure-pipelines.yml

Esta definición de compilación usa una serie de tareas integradas que hacen que la creación de compilaciones sea tan sencilla como construir un conjunto Lego (más sencillo que el gigante Millennium Falcon). Por ejemplo, la tarea NuGet restaura paquetes NuGet, mientras que la tarea VSBuild llama a las herramientas de compilación de Visual Studio para realizar la compilación real. Hay cientos de tareas diferentes disponibles en Azure DevOps, con miles más que mantiene la comunidad. Es probable que, independientemente de las tareas de compilación que quiera ejecutar, alguien ya ha creado una.

Las compilaciones se pueden desencadenar manualmente, mediante una comprobación, según una programación o mediante la finalización de otra compilación. En la mayoría de los casos, conviene compilar en cada comprobación. Las compilaciones se pueden filtrar para que las distintas compilaciones se ejecuten en diferentes partes del repositorio o en ramas diferentes. Esto permite escenarios como ejecutar compilaciones rápidas con pruebas reducidas en las solicitudes de pull y ejecutar un conjunto de regresión completo en el tronco de manera nocturna.

El resultado final de una compilación es una colección de archivos conocidos como artefactos de compilación. Estos artefactos se pueden pasar al paso siguiente del proceso de compilación o agregarlos a una fuente de Azure Artifacts, por lo que otras compilaciones pueden consumirlos.

Lanzamientos de Azure DevOps

Las compilaciones se encargan de compilar el software en un paquete que puede enviarse, pero los artefactos se deben seguir insertando en un entorno de prueba para completar la entrega continua. Para ello, Azure DevOps usa una herramienta independiente denominada Releases. La herramienta Releases usa la misma biblioteca de tareas que estaban disponibles para la compilación, pero presenta un concepto de "fases". Una fase es un entorno aislado en el que se instala el paquete. Por ejemplo, un producto podría usar un desarrollo, un control de calidad y un entorno de producción. El código se entrega continuamente en el entorno de desarrollo donde se pueden ejecutar pruebas automatizadas en él. Una vez que esas pruebas se superen, la versión se mueve al entorno de aseguramiento de calidad para las pruebas manuales. Por último, el código se lanza a producción donde es visible para todos.

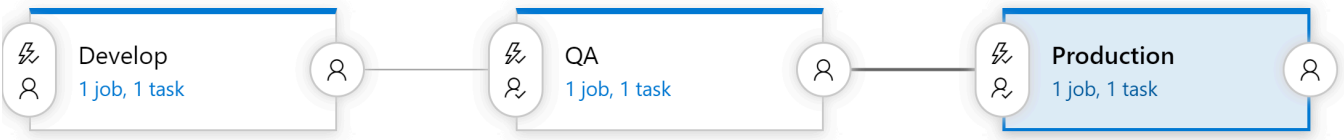


Figura 10-10 - Canalización de lanzamiento

Cada fase de la compilación se puede desencadenar automáticamente mediante la finalización de la fase anterior. Sin embargo, en muchos casos, esto no es deseable. Mover código a

producción podría requerir aprobación de alguien. La herramienta de lanzamientos facilita esto permitiendo la intervención de los aprobadores en cada paso del proceso de lanzamiento. Las reglas se pueden configurar de manera que una persona o grupo específico deba aprobar una versión antes de que pase a producción. Estas puertas permiten comprobaciones manuales de calidad y también para el cumplimiento de los requisitos normativos relacionados con el control de lo que entra en producción.

Canalizaciones de compilación para todos

No hay ningún costo para configurar muchas canalizaciones de compilación, por lo que es ventajoso tener al menos una canalización de compilación por microservicio. Idealmente, los microservicios se pueden implementar de forma independiente en cualquier entorno, por lo que cada uno de ellos puede liberarse a través de su propia canalización sin liberar una masa de código no relacionado es perfecto. Cada canalización puede tener su propio conjunto de aprobaciones, lo que permite variaciones en el proceso de construcción para cada servicio.

Control de versiones

Un inconveniente de usar la funcionalidad Releases es que no se puede definir en un archivo protegido `azure-pipelines.yml` . Hay muchas razones por las que podría querer hacerlo, como tener definiciones de versión para cada rama o incluir un esquema de versión en la plantilla de su proyecto. Por suerte, estamos trabajando para trasladar algunas de las fases de compatibilidad al componente de compilación. Esto se conoce como compilación de varias fases y la primera versión ya está disponible.

Siguiente anterior

Marcas de características

18/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

En el capítulo 1, afirmamos que ser nativo en la nube se trata fundamentalmente de la velocidad y la agilidad. Los usuarios esperan una capacidad de respuesta rápida, características innovadoras y un tiempo de inactividad cero. `Feature flags` son una técnica de implementación moderna que ayuda a aumentar la agilidad de las aplicaciones nativas de la nube. Permiten implementar nuevas características en un entorno de producción, pero restringen su disponibilidad. Con solo accionar un interruptor, puedes activar una nueva funcionalidad para ciertos usuarios sin reiniciar la aplicación ni desplegar nuevo código. Distinguen el lanzamiento de las nuevas características de su implementación de código.

Las banderas de funcionalidades se basan en la lógica condicional que controla la visibilidad de la funcionalidad para usuarios en tiempo de ejecución. En los sistemas nativos de nube modernos, es habitual implementar nuevas características en producción pronto, pero probarlas con un público limitado. A medida que aumenta la confianza, la característica se puede implementar incrementalmente en audiencias más amplias.

Otros casos de uso de las marcas de características incluyen:

- Restrinja la funcionalidad premium a grupos de clientes específicos dispuestos a pagar tarifas de suscripción más altas.
- Para estabilizar un sistema, desactive rápidamente una característica problemática, evitando los riesgos de una reversión o corrección urgente.

- Deshabilite una característica opcional con un consumo elevado de recursos durante los períodos de uso máximo.
- Llevar a cabo `experimental feature releases` en segmentos de usuario pequeños para validar la viabilidad y popularidad.

Las marcas de características también promueven el desarrollo `trunk-based`. Es un modelo de bifurcación de control de código fuente en el que los desarrolladores colaboran en características de una sola rama. El enfoque minimiza el riesgo y la complejidad de combinar un gran número de ramas de características de larga duración. Las características no están disponibles hasta que se activan.

Implementación de banderas de funcionalidades

En su esencia, una bandera de características es una referencia a un simple `decision object`. Devuelve un estado booleano de `on` o `off`. La bandera suele envolver un bloque de código que encapsula una característica. El estado de la marca determina si ese bloque de código se ejecuta para un usuario determinado. En la figura 10-11 se muestra la implementación.

C#

```
if (featureFlag) {
    // Run this code block if the featureFlag value is true
} else {
    // Run this code block if the featureFlag value is false
}
```

Figura 10-11 - Implementación sencilla de la bandera de característica.

Tenga en cuenta cómo este enfoque separa la lógica de decisión del código de característica.

En el capítulo 1, analizamos `Twelve-Factor App`. Las recomendaciones aconsejan mantener los valores de configuración externos al código ejecutable de la aplicación. Cuando sea necesario, la configuración se puede leer desde la fuente externa. Los valores de configuración de la marca de características también deben ser independientes de su código base. Al externalizar la configuración de la marca en un repositorio independiente, puede cambiar el estado de la marca sin modificar ni volver a implementar la aplicación.

Azure App Configuration proporciona un repositorio centralizado para las marcas de características. Con él, se definen diferentes tipos de marcas de características y se manipulan sus estados de forma rápida y segura. Agregue las bibliotecas cliente de App Configuration a su aplicación para habilitar la funcionalidad de las banderas de características. Se admiten varios marcos de lenguaje de programación.

Las marcas de características se pueden implementar fácilmente en un servicio ASP.NET Core. La instalación de las bibliotecas de administración de características de .NET y el proveedor de App Configuration le permiten agregar marcas de características mediante declaración al código. Habilitan `FeatureGate` atributos para que no tengas que escribir manualmente sentencias if en el código base.

Una vez configurado en la clase Startup, puede agregar la funcionalidad de marca de características en el nivel de controlador, acción o middleware. La figura 10-12 presenta la implementación del controlador y la acción:

C#

```
[FeatureGate(MyFeatureFlags.FeatureA)]
public class ProductController : Controller
{
    ...
}
```

C#

```
[FeatureGate(MyFeatureFlags.FeatureA)]
public IActionResult UpdateProductStatus()
{
    return ObjectResult(ProductDto);
}
```

Figura 10-12 - Implementación de la marca de características en un controlador y una acción.

Si se deshabilita una marca de característica, el usuario recibirá un código de estado 404 (no encontrado) sin cuerpo de respuesta.

Las marcas de características también se pueden insertar directamente en clases de C#. En la figura 10-13 se muestra la inserción de marcas de características:

C#

```
public class ProductController : Controller
{
    private readonly IFeatureManager _featureManager;

    public ProductController(IFeatureManager featureManager)
    {
        _featureManager = featureManager;
    }
}
```

Figura 10-13 : inserción de marcas de características en una clase.

Las bibliotecas de Administración de características administran el ciclo de vida de la marca de características en segundo plano. Por ejemplo, para minimizar un gran número de llamadas al almacén de configuración, las bibliotecas almacenan en caché los estados de las banderas durante una duración especificada. Pueden garantizar la inmutabilidad de los estados de marca durante una llamada de solicitud. También ofrecen un `Point-in-time snapshot`. Puede reconstruir el historial de cualquier clave-valor y proporcionar su valor pasado en cualquier momento dentro de los siete días anteriores.

Siguiente anterior

Infraestructura como código

Artículo • 10/05/2023

Sugerencia

Este contenido es un extracto del libro electrónico "Architecting Cloud Native .NET Applications for Azure" (Diseño de la arquitectura de aplicaciones .NET nativas en la nube para Azure), disponible en **Documentos de .NET** o como un PDF descargable y gratuito que se puede leer sin conexión.

Descargar PDF

Los sistemas nativos en la nube adoptan microservicios, contenedores y diseño moderno del sistema para lograr velocidad y agilidad. Proporcionan fases automatizadas de compilación y versión para garantizar código coherente y de calidad. Pero eso es solo parte de la historia. ¿Cómo se aprovisionan los entornos en la nube en los que se ejecutan estos sistemas?

Las aplicaciones nativas de nube modernas adoptan la práctica ampliamente aceptada de infraestructura como código o `IaC`. Con IaC, automatiza el aprovisionamiento de plataformas. Básicamente, aplica prácticas de ingeniería de software como pruebas y control de versiones a las prácticas de DevOps. La infraestructura y las implementaciones son automatizadas, coherentes y repetibles. Al igual que la entrega continua automatiza el modelo tradicional de implementaciones manuales, la infraestructura como código (IaC) está evolucionando cómo se administran los entornos de aplicación.

Las herramientas como Azure Resource Manager (ARM), Terraform y la interfaz de la línea de comandos (CLI) de Azure permiten crear scripts declarativos de la infraestructura en la nube que necesite.

Plantillas del Administrador de recursos de Azure

ARM significa Azure Resource Manager. Es un motor de aprovisionamiento de API integrado en Azure y expuesto como un servicio de API. ARM le permite implementar, actualizar, eliminar y administrar los recursos contenidos en el grupo de recursos de Azure en una sola operación coordinada. El motor se proporciona con una plantilla basada en JSON que especifica los recursos que necesita y su configuración. ARM organiza automáticamente la implementación en el orden correcto respetando las dependencias. El motor garantiza la idempotencia. Si ya existe un recurso deseado con la misma configuración, se omitirá el aprovisionamiento.

Las plantillas de Azure Resource Manager son un lenguaje basado en JSON para definir varios recursos en Azure. El esquema básico tiene un aspecto similar al de la figura 10-14.

JSON

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
  "contentVersion": "",
  "apiProfile": "",
  "parameters": {  },
  "variables": {  },
  "functions": [  ],
  "resources": [  ],
  "outputs": {  }
}
```

Figura 10-14: esquema de una plantilla de Resource Manager

Dentro de esta plantilla, puede definir un contenedor de almacenamiento dentro de la sección de recursos de la siguiente manera:

JSON

```
"resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "apiVersion": "2018-07-01",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2",
```

```
      "properties": {}
    }
  ],
```

Figura 10-15: ejemplo de una cuenta de almacenamiento definida en una plantilla de Resource Manager

Una plantilla de ARM se puede parametrizar con información de configuración y entorno dinámico. Al hacerlo, se puede reutilizar para definir diferentes entornos, como desarrollo, control de calidad o producción. Normalmente, la plantilla crea todos los recursos dentro de un único grupo de recursos de Azure. Es posible definir varios grupos de recursos en una sola plantilla de Resource Manager, si es necesario. Para eliminar todos los recursos de un entorno, elimine el propio grupo de recursos. El análisis de costos también se puede ejecutar en el nivel de grupo de recursos, lo que permite realizar una contabilidad rápida de cuánto cuesta cada entorno.

Hay muchos ejemplos de plantillas de ARM disponibles en el proyecto Plantillas de inicio rápido de Azure en GitHub. Pueden ayudar a acelerar la creación de una plantilla o la modificación de una existente.

Las plantillas de Resource Manager se pueden ejecutar de muchas maneras. Tal vez la manera más sencilla es simplemente pegarlas en Azure Portal. En el caso de las implementaciones experimentales, este método puede ser rápido. También se pueden ejecutar como parte de un proceso de compilación o versión en Azure DevOps. Hay tareas que aprovecharán las conexiones en Azure para ejecutar las plantillas. Los cambios en las plantillas de Resource Manager se aplican de forma incremental, lo que significa que para agregar un nuevo recurso solo es necesario agregarlo a la plantilla. Las herramientas conciliarán las diferencias entre los recursos actuales y los definidos en la plantilla. Los recursos se crearán o modificarán para que coincidan con lo que se define en la plantilla.

Terraform

A menudo, las aplicaciones nativas de la nube se construyen para que sean `cloud agnostic`. Ser así significa que la aplicación no está estrechamente acoplada a un proveedor de nube determinado y se puede implementar en cualquier nube pública.

Terraform es una herramienta de plantillas comercial que puede aprovisionar aplicaciones nativas de la nube en todos los principales actores de la nube: Azure, Google Cloud Platform, AWS y AliCloud. En lugar de usar JSON como lenguaje de definición de plantilla, usa el HCL (HashiCorp Configuration Language) ligeramente más terso.

Un archivo de Terraform de ejemplo que hace lo mismo que la plantilla de Resource Manager anterior (figura 10-15) se muestra en la figura 10-16:

Terraform

```
provider "azurerm" {
  version = "=1.28.0"
}

resource "azurerm_resource_group" "testrg" {
  name     = "production"
  location = "West US"
}

resource "azurerm_storage_account" "testsa" {
  name                     = "${var.storageAccountName}"
  resource_group_name      = "${azurerm_resource_group.testrg.name}"
  location                 = "${var.region}"
  account_tier             = "${var.tier}"
  account_replication_type = "${var.replicationType}"

}
```

Figura 10-16: ejemplo de una plantilla de Resource Manager

Terraform también proporciona mensajes de error intuitivos para las plantillas de problema. También hay una tarea de validación práctica que se puede usar en la fase de compilación para detectar los errores de plantilla al principio.

Al igual que con las plantillas de Resource Manager, las herramientas de línea de comandos están disponibles para implementar plantillas de Terraform. También hay tareas creadas por la comunidad en Azure Pipelines que pueden validar y aplicar plantillas de Terraform.

A veces, las plantillas de Terraform y ARM generan valores significativos, como una cadena de conexión a una base de datos recién creada. Esta información se puede capturar en la canalización de compilación y usarse en tareas posteriores.

Scripts y tareas de la CLI de Azure

Por último, puede aprovechar la CLI de Azure para crear scripts declarativos de la infraestructura en la nube. Los scripts de la CLI de Azure se pueden crear, encontrar y compartir para aprovisionar y configurar casi cualquier recurso de Azure. La CLI es fácil de usar con una curva de aprendizaje suave. Los scripts se ejecutan en PowerShell o Bash. También son sencillos de depurar, especialmente cuando se comparan con las plantillas de ARM.

Los scripts de la CLI de Azure funcionan bien cuando es necesario anular y volver a implementar la infraestructura. La actualización de un entorno existente puede resultar complicada. Muchos comandos de la CLI no son idempotentes. Esto significa que volverán a crear el recurso cada vez que se ejecuten, incluso si el recurso ya existe. Siempre es posible agregar código que compruebe la existencia de cada recurso antes de crearlo. Pero, al hacerlo, el script puede hincharse y ser difícil de administrar.

Estos scripts también se pueden insertar en canalizaciones de Azure DevOps como `Azure CLI tasks`. La ejecución de la canalización invoca el script.

En la figura 10-17 se muestra un fragmento de código YAML que enumera la versión de la CLI de Azure y los detalles de la suscripción. Tenga en cuenta cómo se incluyen los comandos de la CLI de Azure en un script insertado.

YAML

```yaml
- task: AzureCLI@2
  displayName: Azure CLI
  inputs:
    azureSubscription: <Name of the Azure Resource Manager service
connection>
    scriptType: ps
    scriptLocation: inlineScript
    inlineScript: |
      az --version
      az account show
```

Figura 10-17: script de la CLI de Azure

En el artículo ¿Qué es la infraestructura como código?, su autor Sam Guckenheimer describe cómo los equipos que implementan IaC pueden ofrecer entornos estables rápidamente y a escala. Los equipos evitan la configuración manual de entornos y aplican la coherencia mediante la representación del estado deseado de sus entornos a través del código. Las implementaciones de infraestructura con IaC son repetibles y evitan problemas en tiempo de ejecución causados por el desviado de configuración o dependencias que faltan. Los equipos de DevOps pueden trabajar junto con un conjunto unificado de prácticas y herramientas para ofrecer aplicaciones y su infraestructura de apoyo de forma rápida, confiable y a escala".

Agrupaciones de aplicaciones nativas en la nube

18/06/2025

Sugerencia

Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.

Descargar PDF

Una propiedad clave de las aplicaciones nativas de la nube es que aprovechan las funcionalidades de la nube para acelerar el desarrollo. Este diseño suele significar que una aplicación completa usa diferentes tipos de tecnologías. Las aplicaciones se pueden enviar en contenedores de Docker, algunos servicios pueden usar Azure Functions, mientras que otras partes se pueden ejecutar directamente en máquinas virtuales asignadas en servidores metálicos grandes con aceleración de GPU de hardware. No hay dos aplicaciones nativas en la nube iguales, por lo que ha sido difícil proporcionar un único mecanismo para enviarlos.

Los contenedores de Docker se pueden ejecutar en Kubernetes mediante un gráfico de Helm para la implementación. Azure Functions se puede asignar mediante plantillas de Terraform. Por último, las máquinas virtuales se pueden asignar mediante Terraform, pero creadas con Ansible. Esta es una gran variedad de tecnologías y no ha habido ninguna manera de empaquetarlos todos juntos en un paquete razonable. Hasta ahora.

Las agrupaciones de aplicaciones nativas en la nube (CNAB) son un esfuerzo conjunto de muchas empresas de mente comunidad, como Microsoft, Docker y HashiCorp, para desarrollar una especificación para empaquetar aplicaciones distribuidas.

El esfuerzo se anunció en diciembre de 2018, por lo que todavía queda bastante trabajo por hacer para dar a conocer el esfuerzo a la comunidad en general. Sin embargo, ya hay una

especificación abierta y una implementación de referencia conocida como Duffle. Esta herramienta, escrita en Go, es un esfuerzo conjunto entre Docker y Microsoft.

Los CNAB pueden contener diferentes tipos de tecnologías de instalación. Este aspecto permite que coexistan elementos como gráficos de Helm, plantillas de Terraform y cuadernos de estrategias de Ansible en el mismo paquete. Una vez construidos, los paquetes son independientes y portátiles; se pueden instalar desde un stick USB. Los paquetes están firmados criptográficamente para asegurarse de que se originan en la entidad que reclaman.

El núcleo de un CNAB es un archivo denominado `bundle.json`. Este archivo define el contenido de la agrupación, ya sea Terraform o imágenes o cualquier otra cosa. En la figura 11-9 se define un CNAB que invoca a Terraform. Sin embargo, tenga en cuenta que en realidad se define una imagen de invocación que se usa para invocar a Terraform. Cuando se empaqueta, el archivo de Docker que se encuentra en el directorio *cnab* se integra en una imagen de Docker, que se incluirá en el paquete. Tener Terraform instalado dentro de un contenedor de Docker en el paquete significa que los usuarios no necesitan tener Terraform instalado en su máquina para ejecutar la agrupación.

JSON

```
{
    "name": "terraform",
    "version": "0.1.0",
    "schemaVersion": "v1.0.0-WD",
    "parameters": {
        "backend": {
            "type": "boolean",
            "defaultValue": false,
            "destination": {
                "env": "TF_VAR_backend"
            }
        }
    },
    "invocationImages": [
        {
        "imageType": "docker",
        "image": "cnab/terraform:latest"
        }
    ],
    "credentials": {
        "tenant_id": {
            "env": "TF_VAR_tenant_id"
        },
        "client_id": {
            "env": "TF_VAR_client_id"
        },
        "client_secret": {
            "env": "TF_VAR_client_secret"
        },
```

```
        "subscription_id": {
            "env": "TF_VAR_subscription_id"
        },
        "ssh_authorized_key": {
            "env": "TF_VAR_ssh_authorized_key"
        }
    },
    "actions": {
        "status": {
            "modifies": true
        }
    }
}
```

Figura 10-18 : Un archivo de Terraform de ejemplo

`bundle.json` también define un conjunto de parámetros que se pasan a Terraform. La parametrización de la agrupación permite la instalación en varios entornos diferentes.

El formato CNAB también es flexible, lo que permite su uso en cualquier nube. Incluso se puede usar en soluciones locales como OpenStack.

Decisiones de DevOps

Hay tantas herramientas excelentes en el espacio de DevOps en estos días e incluso libros y documentos fantásticos sobre cómo tener éxito. Un libro favorito para empezar a trabajar en el recorrido de DevOps es The Phoenix Project, que sigue la transformación de una empresa ficticia de NoOps a DevOps. Una cosa es segura: DevOps ya no es algo deseable al implementar aplicaciones complejas de la nube nativas. Es un requisito y debe planearse y tener recursos al principio de cualquier proyecto.

Referencias

- Azure DevOps
- Administrador de Recursos de Azure
- Terraform
- Azure CLI

Siguiente anterior

Resumen: Diseño de aplicaciones nativas de nube

19/06/2025

> **Sugerencia**
>
> Este contenido es un extracto del libro electrónico, Arquitectura de aplicaciones .NET nativas de nube para Azure, disponible en .NET Docs o como un PDF descargable gratuito que se puede leer sin conexión.
>
> Descargar PDF
>
>
>

En resumen, estas son conclusiones importantes de esta guía:

- **Nativo en la nube** consiste en diseñar aplicaciones modernas que permiten adaptarse a cambios rápidos, funcionar a gran escala y ser resilientes, en entornos modernos y dinámicos como nubes públicas, privadas e híbridas.
- Cloud Native Computing Foundation **(CNCF)** es un consorcio influyente de código abierto de más de 300 grandes corporaciones. Es responsable de impulsar la adopción de la informática nativa en la nube en las pilas de tecnología y en la nube.
- **Las directrices de CNCF** recomiendan que las aplicaciones nativas de la nube adopten seis pilares importantes, como se muestra en la figura 11-1:

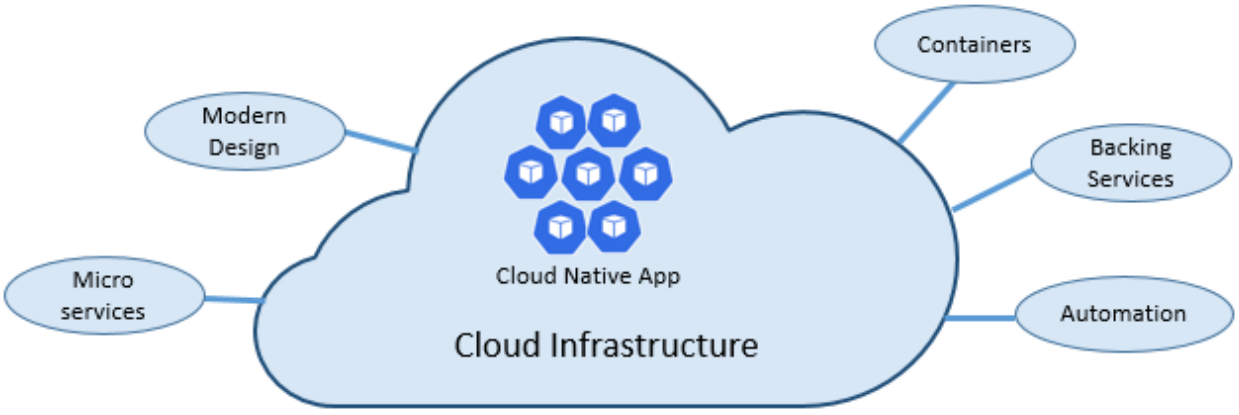


Figura 11-1. Pilares fundamentales nativos de la nube

- Estos pilares nativos de la nube incluyen:
  - La nube y su modelo de servicio subyacente
  - Principios de diseño modernos
  - Microservicios
  - Contenedorización y orquestación de contenedores
  - Servicios de respaldo basados en la nube, como bases de datos y agentes de mensajes
  - Automatización, incluida la infraestructura como código y la implementación de código

- **Kubernetes** es el entorno de hospedaje preferido para la mayoría de las aplicaciones nativas de la nube. Los servicios más pequeños y simples a veces se hospedan en plataformas sin servidor, como Azure Functions. Entre muchas características clave de automatización, ambos entornos proporcionan escalado automático para controlar los volúmenes de cargas de trabajo fluctuantes.

- **La comunicación del servicio** se convierte en una decisión de diseño significativa al construir una aplicación nativa de la nube. Las aplicaciones suelen exponer una puerta de enlace de API para administrar la comunicación de cliente front-end. A continuación, los microservicios de back-end se esfuerzan por comunicarse entre sí mediante la implementación de patrones de comunicación asincrónicos, siempre que sea posible.

- **gRPC** es un marco moderno y de alto rendimiento que evoluciona el protocolo de llamada a procedimientos remotos (RPC) antiguo. Las aplicaciones nativas de la nube suelen adoptar gRPC para simplificar la mensajería entre los servicios back-end. gRPC usa HTTP/2 como protocolo de transporte. Puede ser hasta 8 veces más rápido que la serialización JSON con tamaños de mensaje de 60 a 80% más pequeños. gRPC es de código abierto y está administrado por Cloud Native Computing Foundation (CNCF).

- **Los datos distribuidos** son un modelo que a menudo implementan las aplicaciones nativas de la nube. Las aplicaciones separan la funcionalidad empresarial en microservicios pequeños e independientes. Cada microservicio encapsula sus propias

dependencias, datos y estado. El modelo de base de datos compartido clásico evoluciona en una de las muchas bases de datos más pequeñas, cada una de las cuales se alinea con un microservicio. Cuando se calman las aguas, emergemos con un diseño que expone un modelo `database-per-microservice`.

- **No-SQL las bases de datos** hacen referencia a almacenes de datos no relacionales y de alto rendimiento. Se destacan en sus características de facilidad de uso, escalabilidad, resistencia y disponibilidad. Los servicios de gran volumen que requieren tiempo de respuesta en subsegundos favorecen las bases de datos NoSQL. La proliferación de tecnologías NoSQL para sistemas nativos en la nube distribuidos no puede ser exagerada.

- **NewSQL** es una tecnología emergente de base de datos que combina la escalabilidad distribuida de NoSQL y las garantías ACID de una base de datos relacional. Las bases de datos NewSQL tienen como destino sistemas empresariales que deben procesar grandes volúmenes de datos, entre entornos distribuidos, con cumplimiento transaccional o ACID completo. Cloud Native Computing Foundation (CNCF) cuenta con varios proyectos de base de datos NewSQL.

- **La resistencia** es la capacidad de su sistema de reaccionar ante errores y seguir siendo funcional. Los sistemas nativos de la nube adoptan una arquitectura distribuida en la que el error es inevitable. Las aplicaciones deben construirse para responder elegantemente al error y volver rápidamente a un estado de funcionamiento completo.

- **Las mallas de servicio** son una capa de infraestructura configurable con funcionalidades integradas para controlar la comunicación del servicio y otros desafíos transversales. Desacoplan las responsabilidades transversales del código empresarial. Estas responsabilidades se trasladan a un intermediario de servicio. Conocido como `Sidecar pattern`, el proxy se implementa en un proceso independiente para proporcionar aislamiento del código empresarial.

- **La observabilidad** es una consideración clave de diseño para las aplicaciones nativas de la nube. A medida que los servicios se distribuyen en un clúster de nodos, el registro centralizado, la supervisión y las alertas, son obligatorios. Azure Monitor es una colección de herramientas basadas en la nube diseñadas para proporcionar visibilidad sobre el estado del sistema.

- **La infraestructura como código** es una práctica ampliamente aceptada que automatiza el aprovisionamiento de plataformas. La infraestructura y las implementaciones son automatizadas, coherentes y repetibles. Herramientas como Azure Resource Manager, Terraform y la CLI de Azure, le permiten generar scripts declarativos de la infraestructura en la nube que necesita.

- **La automatización del código** es un requisito para las aplicaciones nativas de la nube. Los sistemas de CI/CD modernos ayudan a cumplir este principio. Proporcionan pasos de compilación e implementación independientes que ayudan a garantizar código coherente y de calidad. La fase de compilación transforma el código en un artefacto binario. La fase de versión recoge el artefacto binario, aplica la configuración del entorno externo e lo implementa en un entorno especificado. Azure DevOps y GitHub son entornos completos de DevOps.

Anterior