(Micro) servicios con notificaciones de eventos

Cengiz Han Seguir
25 de febrero de 2019 · 6 min de lectura

Los microservicios son un estilo arquitectónico que describe el diseño de software como servicios de implementación independiente, débilmente acoplados, que se modelan en torno a un dominio empresarial particular.

La extracción de diferentes capacidades de dominio empresarial de una aplicación monolítica de proceso único y la creación de un diseño de sistema que contenga procesos de servicio más pequeños le permite *escalar* e *implementar* cada servicio por separado.

El hecho de que cada servicio deba implementarse por separado requiere y también permite la automatización de la implementación de aplicaciones, entrega continua.

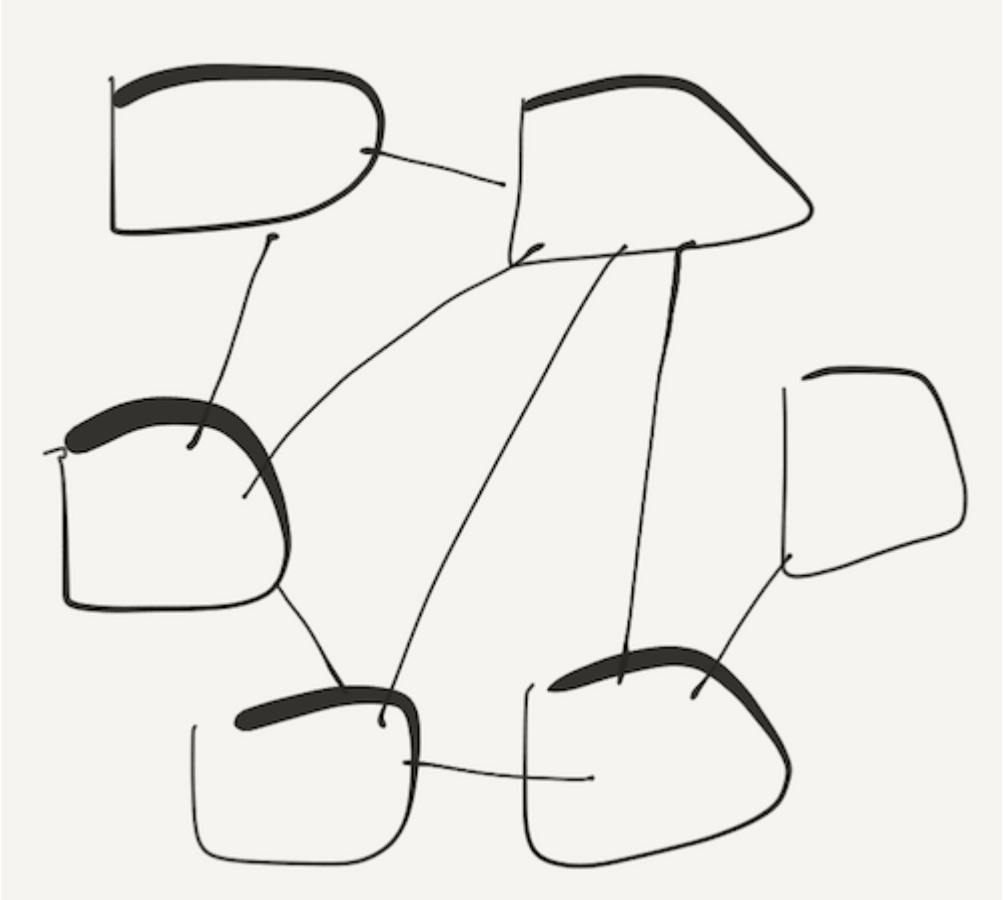
Martin Fowler describe los requisitos previos de los microservicios en sus artículos y enfatiza la importancia de DevOpsCulture .

> *Debe tener esta altura para usar microservicios* .

He trabajado principalmente con microservicios y sistemas controlados por eventos durante casi los últimos 7 años. En esta publicación, intentaré explicar mi punto de vista sobre un desafío particular al diseñar su sistema orientado a servicios.

¿Cómo se hablan entre ellos?

Una de las formas más rápidas con las que las personas comienzan es la comunicación de servicio a servicio entre servicios. Es una forma rápida de comenzar a hacer llamadas de a otro, pero trae muchos problemas. Seguir esta ruta crea un desorden distribuido y trae problemas de que todos los servicios estén disponibles todo el tiempo y cada servicio posiblemente dependa unos de otros, lo que hace que cosas como fallar de manera segura sean realmente difíciles, casi imposibles.



Microservicios llamándose directamente entre sí. Un poco de lío, ¿verdad?

Siempre debe planificar las fallas y cómo va a manejarlas, cómo va a **fallar de manera segura** . La comunicación de sincronización de servicio a servicio hace que el manejo de fallas sea mucho más difícil. Cuando una operación comercial necesita la orquestación de un par de servicios para completar con éxito, debe ***comenzar a pensar en los servicios de desacoplamiento, la consistencia eventual y el contexto limitado*** (una forma de definir los límites de un dominio complejo en el contexto comercial).

> *Un sistema de ser "**fallan - seguras** " significa no es que **el fracaso** es imposible o improbable, sino más bien de evitar o mitigar las consecuencias peligrosas de del sistema de diseño del sistema **falla** . Es decir, si y cuando un "**fracaso - seguro** " sistema "**no** ", es "*

> ***segura*** *" o por lo menos no es menos* ***segura*** *que cuando se estaba operando correctamente. wikipedia .*

Por ejemplo, ¿qué sucede si el servicio SMTP no funciona o si recibe un mensaje de error del Servicio de notificaciones cuando lo llama desde el Servicio de pedidos? ¿Vas a cancelar el pedido por ese motivo? ¡Por supuesto no! Si está realizando una comunicación de servicio a servicio, debe crear un mecanismo de reintento en el contexto de sincronización o mantener un estado en algún lugar que deba ser procesado por un trabajo en ejecución en segundo plano para seguir enviando notificaciones más tarde cuando el servicio de notificación vuelva a estar en línea y en funcionamiento. Y debe pensar en todos los escenarios de falla en todas las comunicaciones de servicio a servicio, no es necesario decir que es una solución compleja, que ***garantiza confiabilidad y capacidad de mantenimiento, y mantener la operatividad simple se*** convierte en algo difícil de lograr.

Arquitectura impulsada por eventos

La arquitectura dirigida por eventos se conoce principalmente como almacenar todos los cambios de estado de la aplicación como secuencia de eventos. Pero cuando observa las diferentes implementaciones en diferentes proyectos, vemos diferentes patrones de uso bajo el nombre de Event Sourcing, Command Query Responsibility Segregation (CQRS), Event Notification. Martin Fowler publicó un artículo y habló sobre diferentes tipos de Arquitectura impulsada por eventos en *abril de 2017 en una Conferencia GoTo* y creó una mejor clasificación para los diferentes modelos que estábamos usando bajo el nombre de Arquitectura impulsada por eventos. No me corresponde clasificarlos cuando un gurú ya lo ha hecho.

He usado CQRS antes en dos proyectos, uno de ellos pasó a producción, uno de ellos se cambió a ***"Eventos como preocupación secundaria"*** como llamé en ese entonces sin saber el mejor nombre para identificar el patrón. Ahora lo sé, se llama ***Notificación de eventos.***

> **Notificación de eventos**: *componentes que se comunican a través de eventos*
> **Transferencia de estado basada en eventos**: *permitiendo que los componentes accedan a los datos sin llamar a la fuente.*
> **Abastecimiento de eventos***: uso de un registro de eventos como registro principal de un sistema*

> **CQRS**: *tener un componente separado para actualizar una tienda de cualquier lector de la tienda*
>
> *Consulte la publicación y el video de Martin en este enlace para obtener más información sobre los diferentes tipos.*

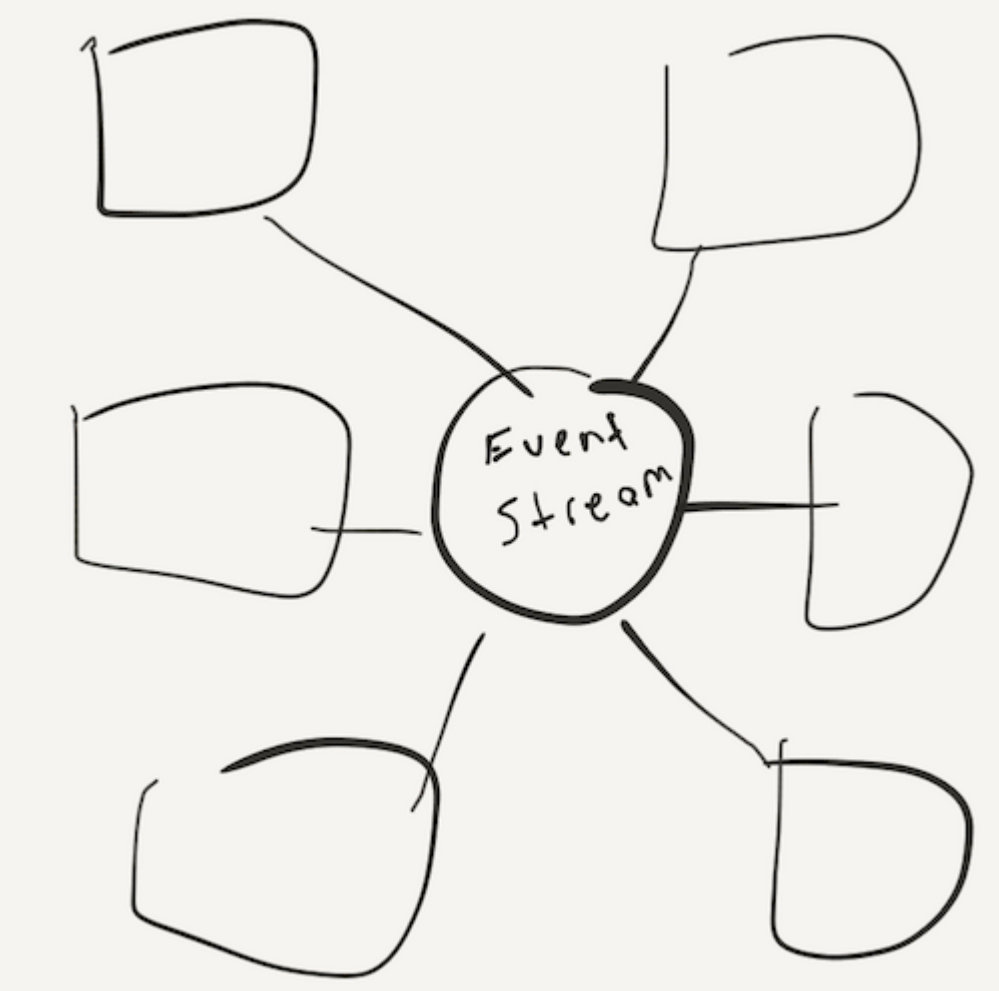
Notificación de eventos

Existe una ligera diferencia entre Notificación de eventos y Transferencia de estado basada en eventos. En *EbST* , los eventos de diseño contienen toda la información para el servicio al consumidor, pero la Notificación de eventos solo contiene información sobre el evento y los servicios al consumidor deben ir y obtener todo el contexto sobre ese evento desde el servicio de origen. Imagine que tiene un evento llamado OrderCreated y ese evento contiene OrderNumber y tal vez algunos otros metadatos sobre el pedido, pero si está creando un Servicio de notificación para enviar un correo electrónico / sms al cliente sobre su pedido reciente, es posible que deba ir al Servicio de pedidos y preguntar detalles de ese pedido con el número de pedido que acaba de recibir a través del evento OrderCreated.

La notificación de eventos proporciona un gran desacoplamiento y permite que otros sistemas se conecten a eventos sin decirle al evento de origen. Puede crear un nuevo servicio que esté interesado en un evento de cualquier servicio sin informar o solicitar ningún cambio en el servicio de origen.

Un problema con este diseño, necesita encontrar una manera de obtener todas las dependencias y qué servicio depende de qué evento de qué servicio. No puede saber qué sucede en todo el sistema cuando ocurre un evento en particular, no puede simplemente leer el código en el servicio fuente y ver qué sucede después de ese evento. Debe revisar todas las suscripciones y averiguar qué sucede en todo el sistema. Este es generalmente el caso que todos ignoramos hasta que terminamos en un lugar del que no tenemos idea de lo que sucede en el sistema.

Una de las cosas adicionales en las que debe pensar una vez que haya comenzado a usar la notificación de eventos es ¿en qué se incluyen los eventos? Si usa Transferencia de estado basada en eventos o necesita llamar al Servicio de pedidos para obtener más información sobre el pedido que acaba de crear.



Los microservicios no se conocen, están desacoplados por notificación de eventos

En el modelo anterior, cuando usa el modelo de notificación de eventos, crea servicios que publican eventos y los servicios posteriores se suscriben a esos eventos en su flujo de eventos. De esta manera, traspasa la complejidad del manejo de fallas al flujo de eventos y a los procesadores de flujo de eventos, sus servicios posteriores.

Order Service publica un evento de pedido creado y el servicio de notificación se suscribe al evento OrderCreated y envía una notificación al cliente.

El sistema de tienda de eventos almacena todos sus eventos, puede usar algo como Kafka, RabbitMQ, nsq para almacenar eventos y enviar cada tipo de evento a los suscriptores. Si un servicio está inactivo temporalmente, la tienda de eventos lo envía al suscriptor cuando regresa, no es necesario implementar nada diferente en este escenario, el diseño de su sistema lo maneja automáticamente.

Ahora tiene una arquitectura impulsada por eventos, que *notifica a* otro sistema cuando sucede algo, cuando tiene un nuevo requisito para reaccionar al evento O *rderCreated* , no necesita ir y hacer ningún cambio en su Servicio de pedidos. Todo lo que necesita hacer es crear un nuevo servicio de suscripción que se suscriba al evento O *rderCreated* .

Digamos que desea mostrar a sus clientes reseñas de productos por separado de otros clientes que realmente compraron ese producto. Son reseñas de clientes verificadas y más valiosas para los compradores potenciales, por lo que crea una nueva suscripción en su Servicio de Reseñas de Productos de Clientes para escuchar O *rderCreated* y otros eventos relacionados con pedidos para marcar a los clientes como clientes verificados para ese producto. Muy simple, no toca la parte existente del sistema, lo construye e implementa por separado sin tocar el servicio upstream.

Estoy planeando escribir una publicación de seguimiento con un ejemplo de código para demostrar lo que mencioné aquí. Hasta entonces, recomiendo encarecidamente ver el discurso de Martin en la conferencia goto .

Publicado originalmente en cengizhan.com el 4 de septiembre de 2017.

Microservicios