MICROSERVICIOS EN NET 5

ARQUITECTURA PARA CONTENEDORES
SESION II

AFORO255 TRAINING CENTER

Instructor

Ivan Cuadros Altamirano
Lead Software Architect

Agenda

- Introducción a la Comunicación en los Microservicios
- Tipos de Comunicación
- Estilos de Comunicación
- Comunicación de HTTP request/response
- Comunicación Asincrónica
- Mediator Pattern
- Manejo de fallas
- Estrategias para manejar fallas parciales
- Circuit Breaker Pattern

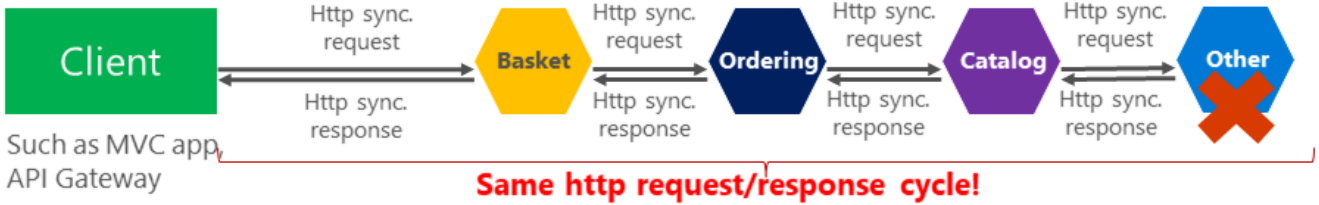
Comunicación entre microservicios

- Una aplicación basada en microservicios es un sistema distribuido que se ejecuta en múltiples procesos o servicios, generalmente incluso en múltiples servidores o hosts. Cada instancia de servicio suele ser un proceso. Por lo tanto, los servicios deben interactuar usando un protocolo de comunicación entre procesos como HTTP, AMQP o un protocolo binario como TCP, dependiendo de la naturaleza de cada servicio.
- Los dos protocolos de uso común son HTTP request/response con las API de recursos (cuando se consulta sobre todo) y la mensajería asincrónica liviana al comunicar actualizaciones a través de múltiples microservicios.

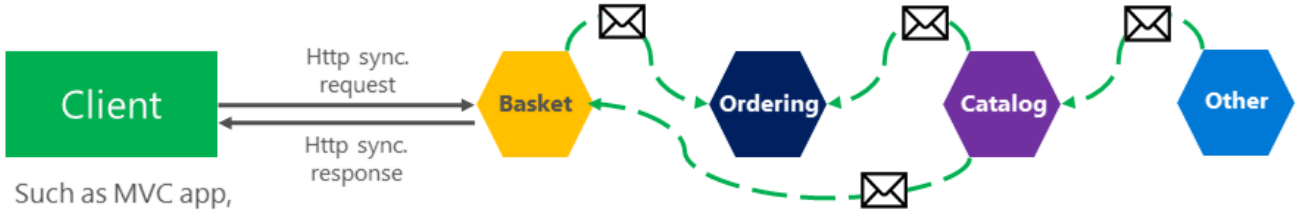
Synchronous vs. async communication across microservices

Anti-pattern

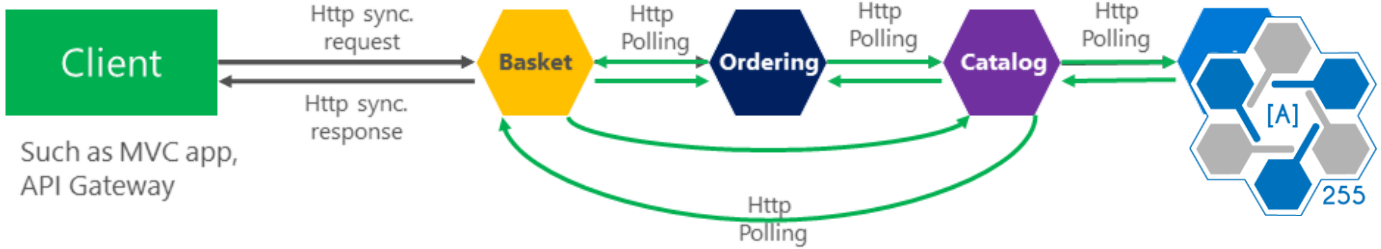
Synchronous
all request/response cycle



Asynchronous
Comm. across internal microservices
(EventBus: like **AMQP**)



"Asynchronous"
Comm. across internal microservices
(Polling: **Http**)



Tipos de comunicación

El cliente y los servicios pueden comunicarse a través de muchos tipos diferentes de comunicación, cada uno dirigido a un escenario y objetivos diferentes. Inicialmente, esos tipos de comunicaciones se pueden clasificar en dos ejes:

- El primer eje define si el protocolo es síncrono o asíncrono
- El segundo eje define si la comunicación tiene un receptor único o múltiples receptores

Tipo de comunicación

El primer eje define si el protocolo es síncrono o asíncrono:

- **Protocolo sincrónico**. HTTP es un protocolo síncrono. El cliente envía una solicitud y espera una respuesta del servicio. Eso es independiente de la ejecución del código del cliente que podría ser síncrono (el subproceso está bloqueado) o asíncrono (el subproceso no está bloqueado y la respuesta eventualmente alcanzará una devolución de llamada). El punto importante aquí es que el protocolo (HTTP / HTTPS) es síncrono y el código del cliente solo puede continuar su tarea cuando recibe la respuesta del servidor HTTP.
- **Protocolo asincrónico**. Otros protocolos como AMQP (un protocolo compatible con muchos sistemas operativos y entornos de nube) usan mensajes asincrónicos. El código del cliente o el remitente del mensaje generalmente no espera una respuesta. Simplemente envía el mensaje como cuando envía un mensaje a una cola RabbitMQ o cualquier otro agente de mensajes.

Tipo de comunicación

El segundo eje define si la comunicación tiene un receptor único o múltiples receptores:

- **Receptor individual.** Cada solicitud debe ser procesada por exactamente un receptor o servicio. Un ejemplo de esta comunicación es el patrón de Comando .
- **Múltiples receptores.** Cada solicitud puede ser procesada por cero a múltiples receptores. Este tipo de comunicación debe ser asíncrono. Un ejemplo es el mecanismo de publicación / suscripción utilizado en patrones como la arquitectura controlada por eventos . Esto se basa en una interfaz de bus de eventos o un intermediario de mensajes cuando se propagan actualizaciones de datos entre múltiples microservicios a través de eventos; Por lo general, se implementa a través de un bus de servicio o un artefacto similar como Azure Service Bus mediante temas y suscripciones .

Tipos de comunicación

Una aplicación basada en microservicios a menudo utilizará una combinación de estos estilos de comunicación. El tipo más común es la comunicación de receptor único con un protocolo síncrono como **HTTP/HTTPS** cuando se invoca un servicio **HTTP API web** normal. Los microservicios también suelen usar protocolos de mensajería para la comunicación asincrónica entre microservicios.

Estilos de comunicación

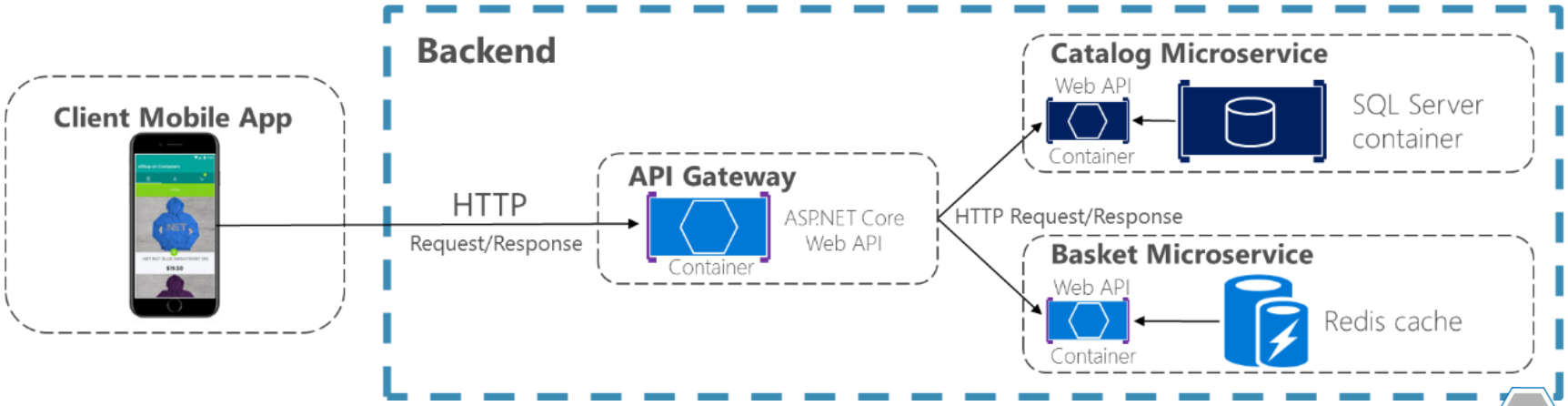
Existen muchos protocolos y opciones que puede usar para la comunicación, según el tipo de comunicación que desee usar. Si está utilizando un mecanismo de comunicación síncrono basado en **request/response**, los protocolos como los enfoques HTTP y REST son los más comunes, especialmente si está publicando sus servicios fuera del host Docker o el clúster de microservicios. Si se está comunicando entre servicios internamente (dentro de su host Docker o clúster de microservicios), es posible que también desee utilizar mecanismos de comunicación de formato binario (como la comunicación remota de Service Fabric o WCF con TCP y formato binario). Alternativamente, puede usar mecanismos de comunicación asíncronos basados en mensajes como AMQP.

Estilos de comunicación

Cuando un cliente utiliza la comunicación de solicitud / respuesta, envía una solicitud a un servicio, luego el servicio procesa la solicitud y devuelve una respuesta. La comunicación de solicitud / respuesta es especialmente adecuada para consultar datos para una IU en tiempo real (una interfaz de usuario en vivo) desde aplicaciones cliente. Por lo tanto, en una arquitectura de microservicio probablemente usará este mecanismo de comunicación para la mayoría de las consultas.

Request/Response Communication for Live Queries and Updates

HTTP and REST based Services



Comunicación asincrónica

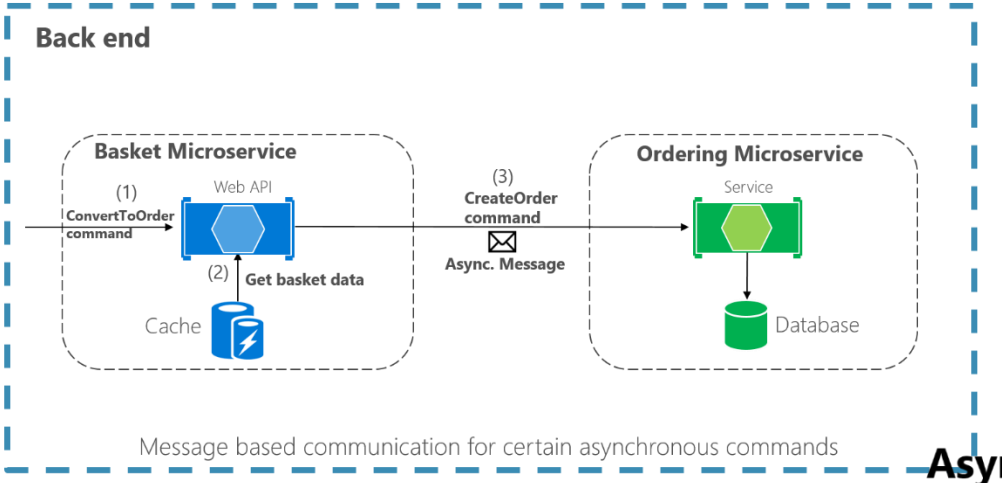
Cuando se usa la mensajería, los procesos se comunican intercambiando mensajes de forma asincrónica. Un cliente realiza un comando o una solicitud a un servicio enviándole un mensaje. Si el servicio necesita responder, envía un mensaje diferente al cliente. Como se trata de una comunicación basada en mensajes, el cliente supone que la respuesta no se recibirá de inmediato y que es posible que no haya ninguna respuesta.

Hay dos tipos de comunicación de mensajes asíncronos:

- Comunicación basada en mensajes de receptor único
- Comunicación basada en mensajes de receptores múltiples.

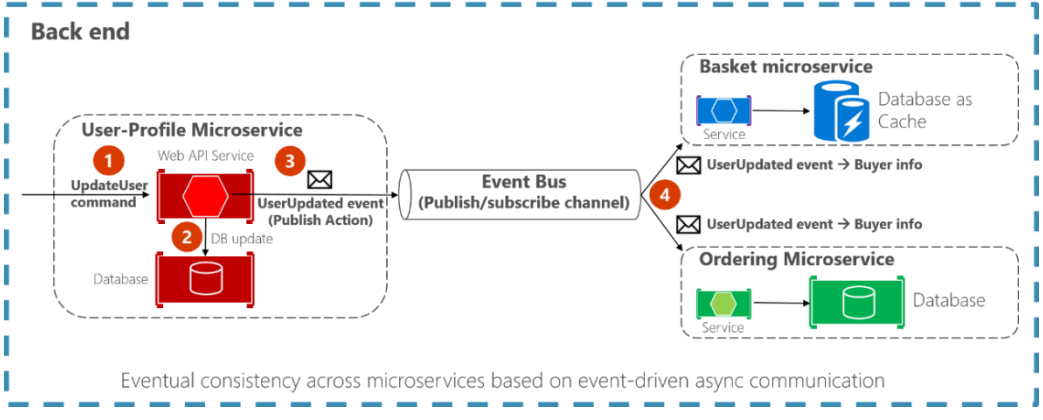
Single receiver message-based communication

(i.e. Message-based Commands)



Asynchronous event-driven communication

Multiple receivers



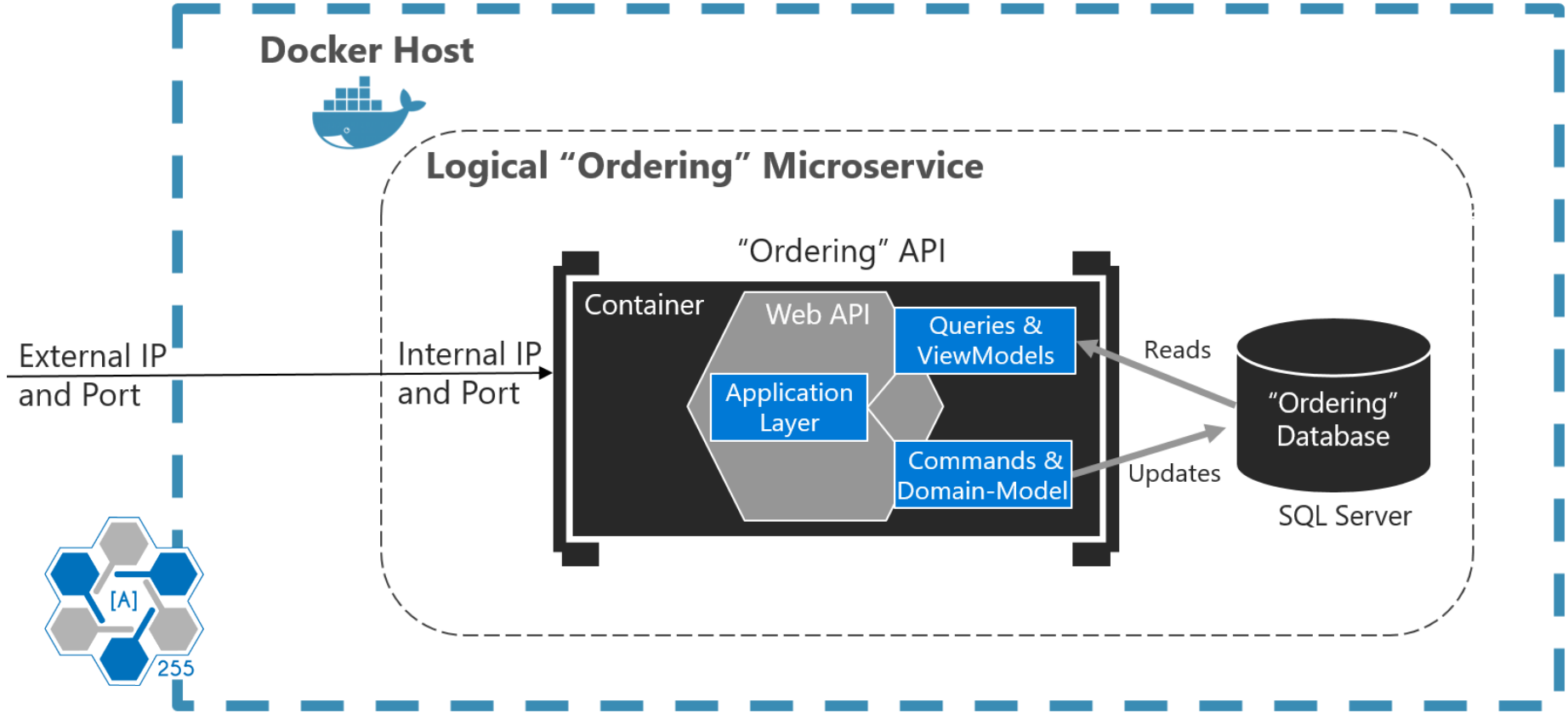
CQRS

CQRS significa segmentación de responsabilidad de consulta de comando. Este es un patrón de diseño o práctica de desarrollo que le permite separar sus operaciones de creación / actualización (lo llamamos - Comandos) de sus lecturas (lo llamamos - Consultas), cada una de las cuales devuelve sus modelos de respuesta para una clara segregación de cada acción.

Separar los comandos y las consultas permite que el modelo de entrada y salida se centre más en la tarea específica que están realizando. Esto también simplifica la prueba de los modelos porque están menos generalizados y, por lo tanto, no están llenos de código adicional.

Simplified CQRS and DDD microservice

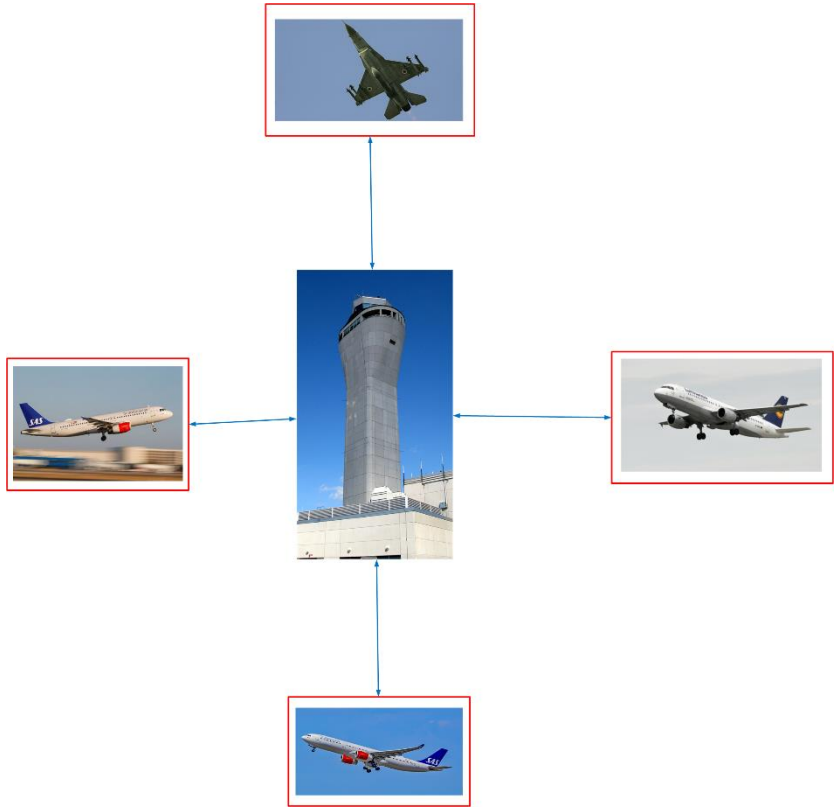
High level design



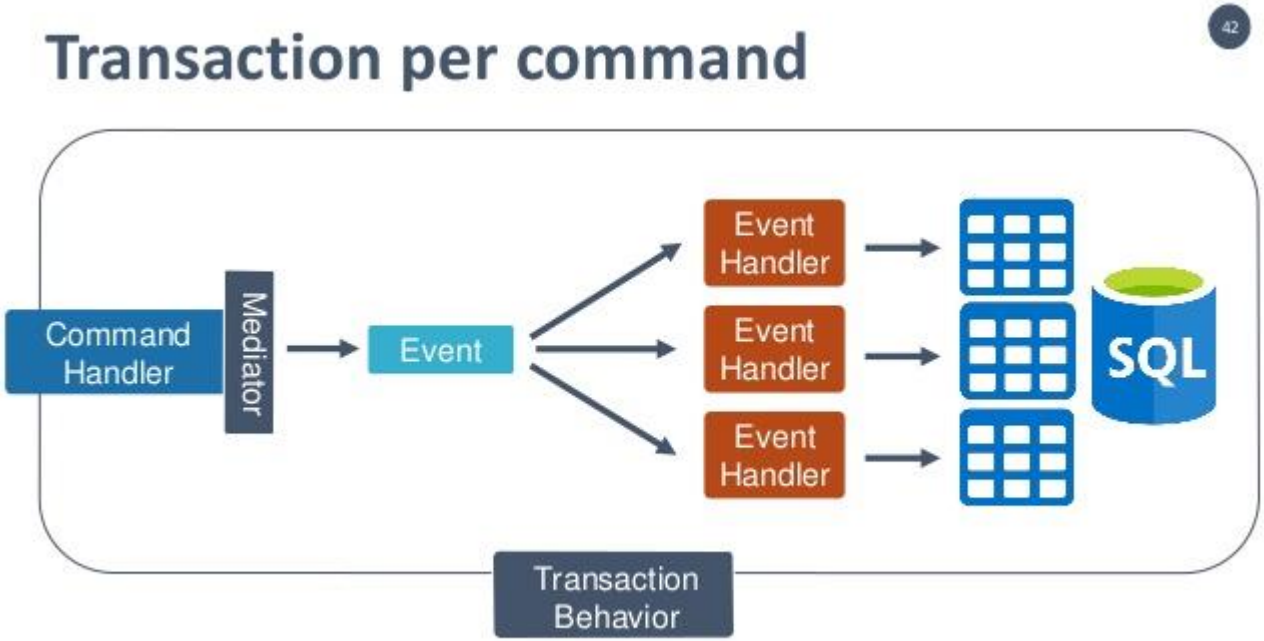
Mediator Pattern

Nos ayuda a resolver los siguientes problemas:

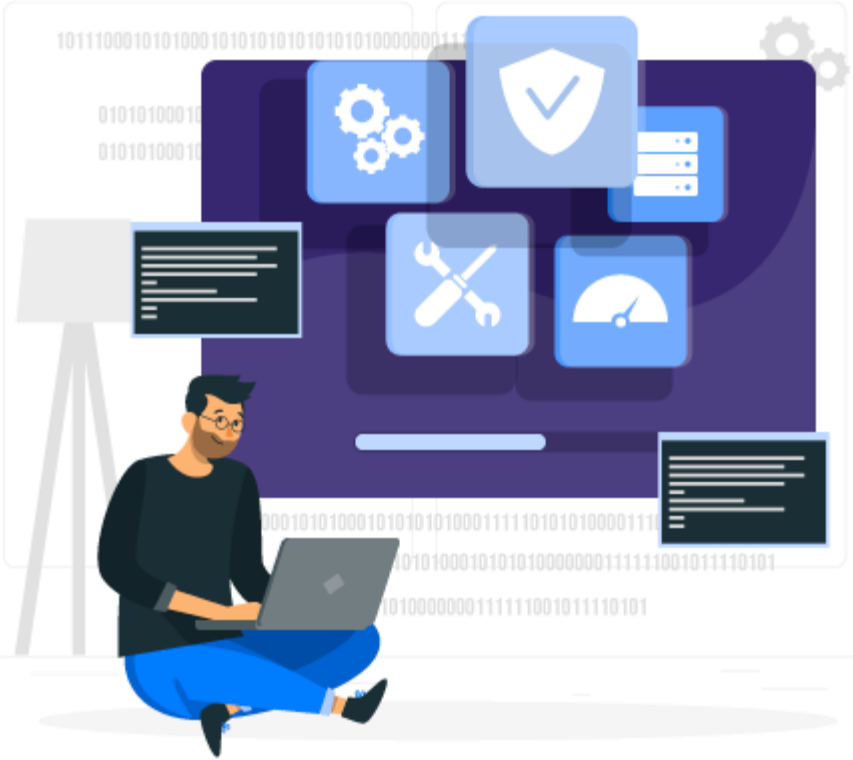
Reducción del número de conexiones entre clases.
Encapsulación de objetos utilizando la interfaz mediadora.
Proporcionando una interfaz unificada para gestionar dependencias entre clases.



CQRS y Mediator Pattern



https://github.com/jbogard/MediatR/wiki/Behaviors



HANDS ON

Manejo de fallas parciales

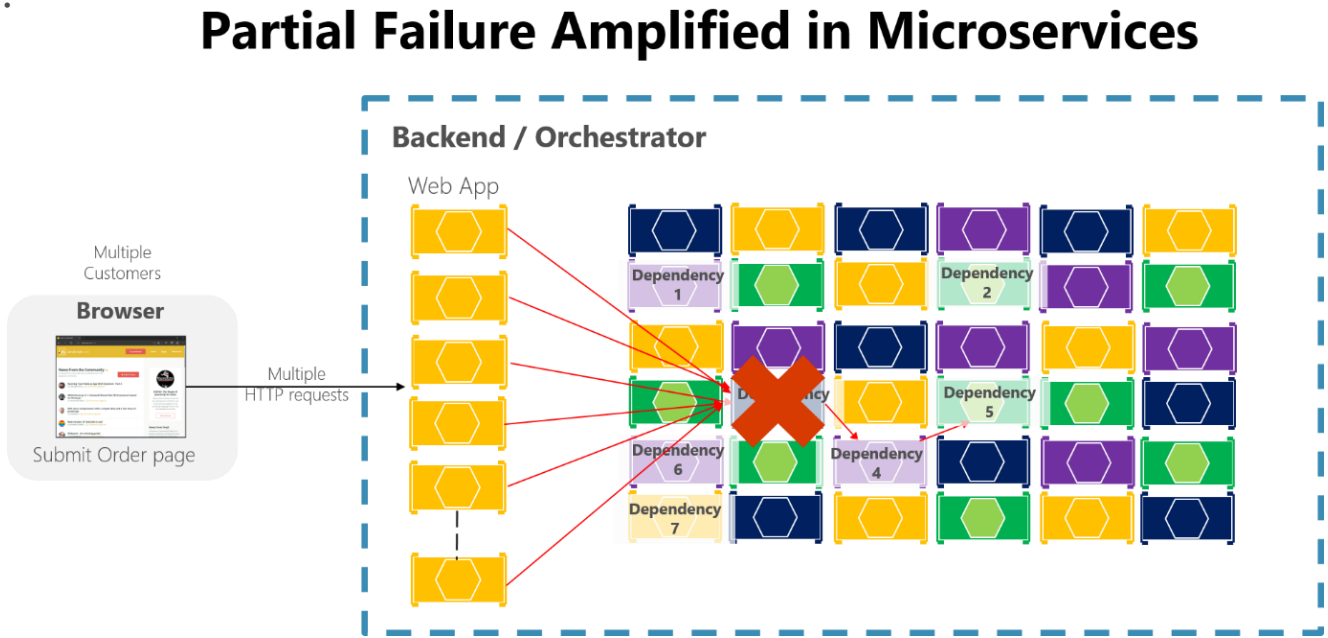
En sistemas distribuidos como aplicaciones basadas en microservicios, existe un riesgo siempre presente de falla parcial.

En una aplicación grande basada en microservicios, cualquier falla parcial puede amplificarse, especialmente si la mayor parte de la interacción interna de microservicios se basa en llamadas HTTP sincrónicas (lo que se considera un antipatrón).

La falla intermitente está prácticamente garantizada en un sistema distribuido y basado en la nube, incluso si cada dependencia tiene una excelente disponibilidad. Esto debería ser un hecho que debe tener en cuenta.

Manejo de fallas parciales

Cuando una dependencia de microservicio falla mientras se maneja un gran volumen de solicitudes, esa falla puede saturar rápidamente todos los hilos de solicitud disponibles en cada servicio y bloquear toda la aplicación.



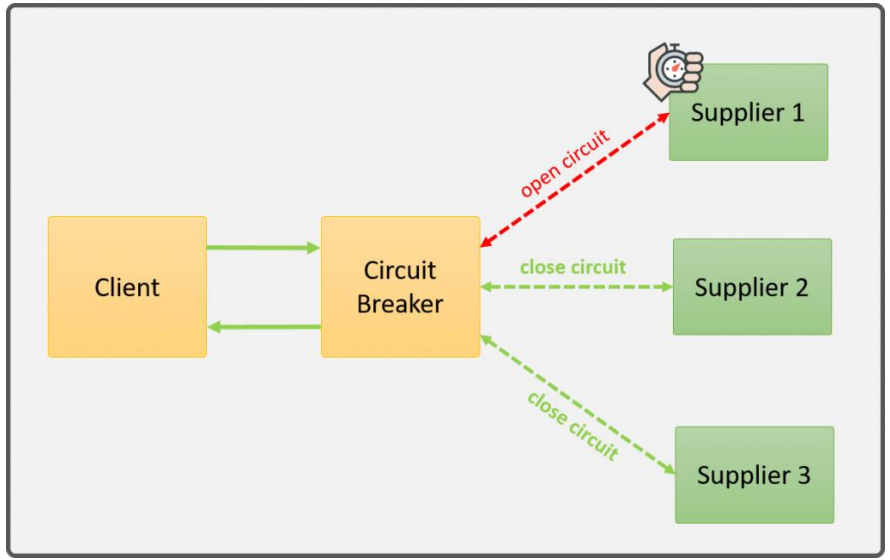
Estrategias para manejar fallas parciales

Las estrategias para lidiar con fallas parciales son las siguientes:

- Utilice la comunicación asincrónica (por ejemplo, comunicación basada en mensajes) a través de microservicios internos
- Usar reintentos con retroceso exponencial
- Evite los tiempos de espera de la red
- Use el patrón de disyuntor
- Proporcionar retrocesos
- Limite el número de solicitudes en cola

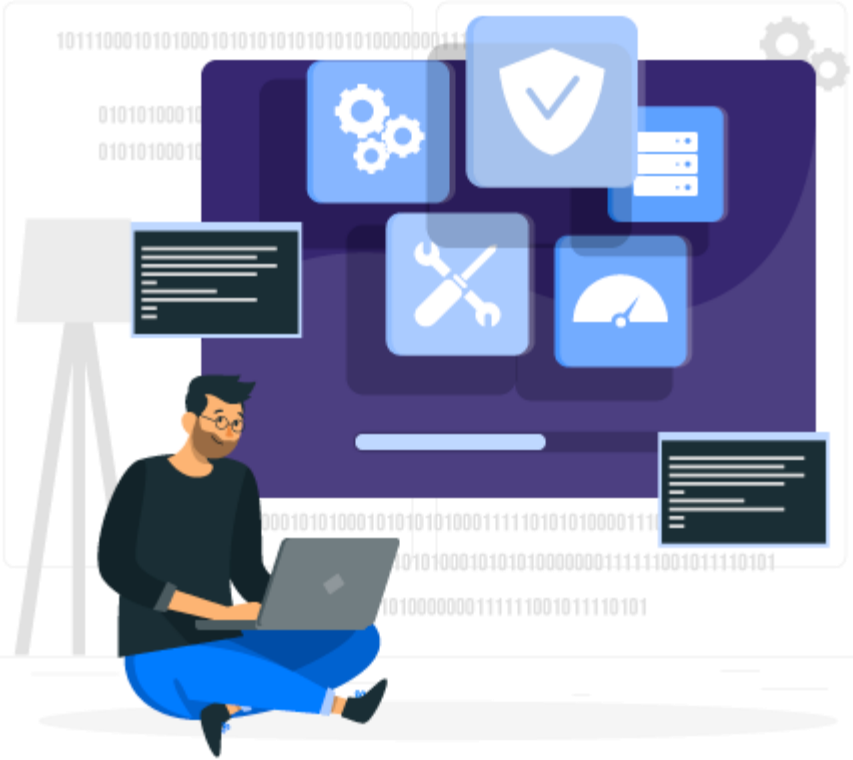
Circuit breaker pattern

Permite manejar las fallas que pueden tardar una cantidad variable de tiempo en recuperarse, como puede ocurrir cuando intenta conectarse a un servicio o recurso remoto. Manejar este tipo de falla puede mejorar la estabilidad y la resistencia de una aplicación.



Circuit breaker pattern

El patrón del disyuntor tiene un propósito diferente al patrón de reintento. El patrón Reintentar permite que una aplicación vuelva a intentar una operación con la expectativa de que la operación finalmente tenga éxito. El patrón del disyuntor impide que una aplicación realice una operación que probablemente falle. Una aplicación puede combinar estos dos patrones utilizando el patrón Reintentar para invocar una operación a través de un interruptor automático. Sin embargo, la lógica de reintento debe ser sensible a las excepciones devueltas por el disyuntor, y debe abandonar los intentos de reintento si el disyuntor indica que una falla no es transitoria.



HANDS ON

¡Muchas gracias!