Sesión 04

Persistencia y consistencia de datos – Parte II

Instructor:

ERICK ARÓSTEGUI

earostegui@galaxy.edu.pe

GALAXY
TRAINING

ÍNDICE

GALAXY
TRAINING

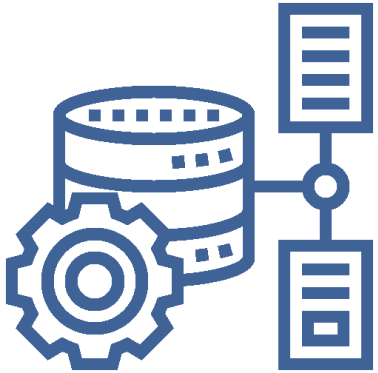
01

Introducción del patrón SAGA.

Transacciones en un sistema distribuido

Consistencia de los datos

- Las transacciones son el enfoque tradicional.

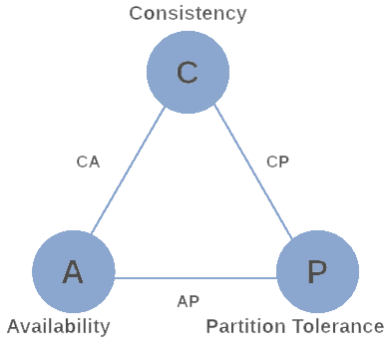


Transacciones del sistema monolítico

- Base de datos única, que es la única verdad

Transacciones de microservicios

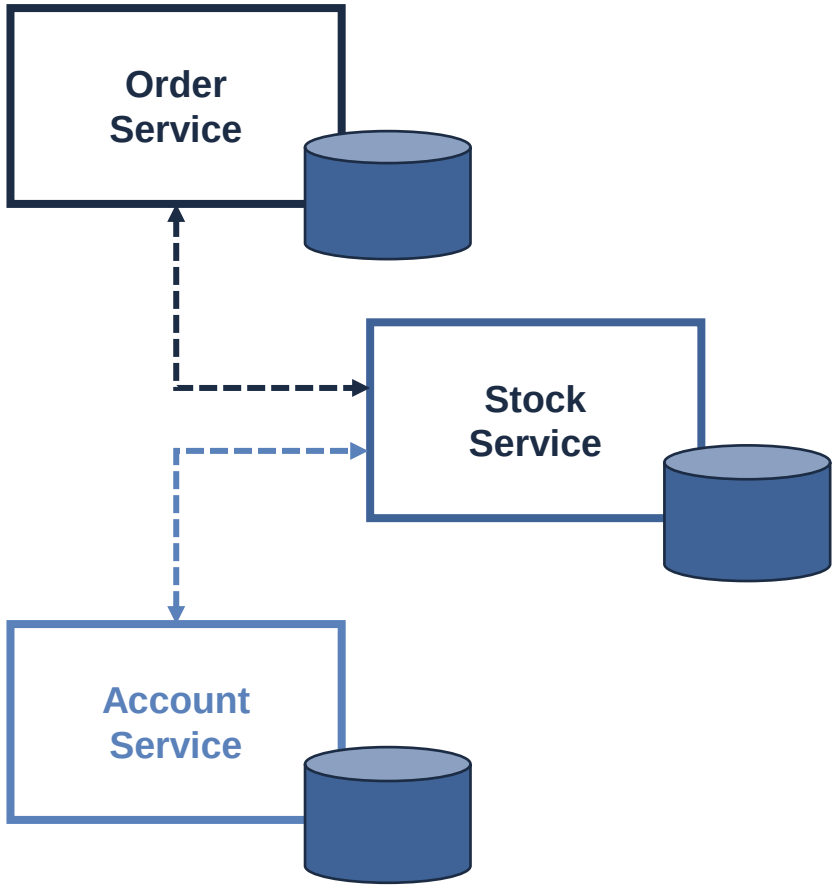
- Arquitectura distribuida
- Datos distribuidos
- Transacciones distribuidas



Teorema de CAP

- La falla de la red sucederá
- Disponibilidad de datos o consistencia de datos?

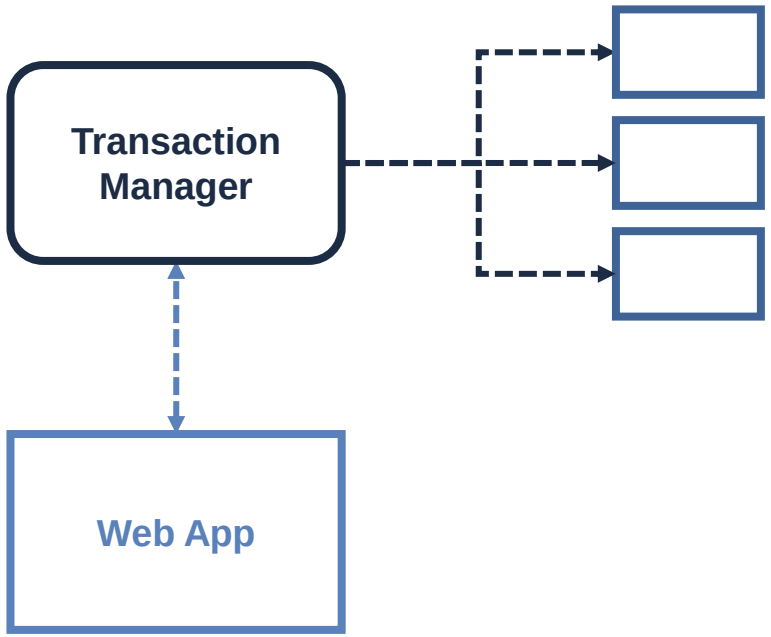
Introducción del patrón SAGA



Opciones

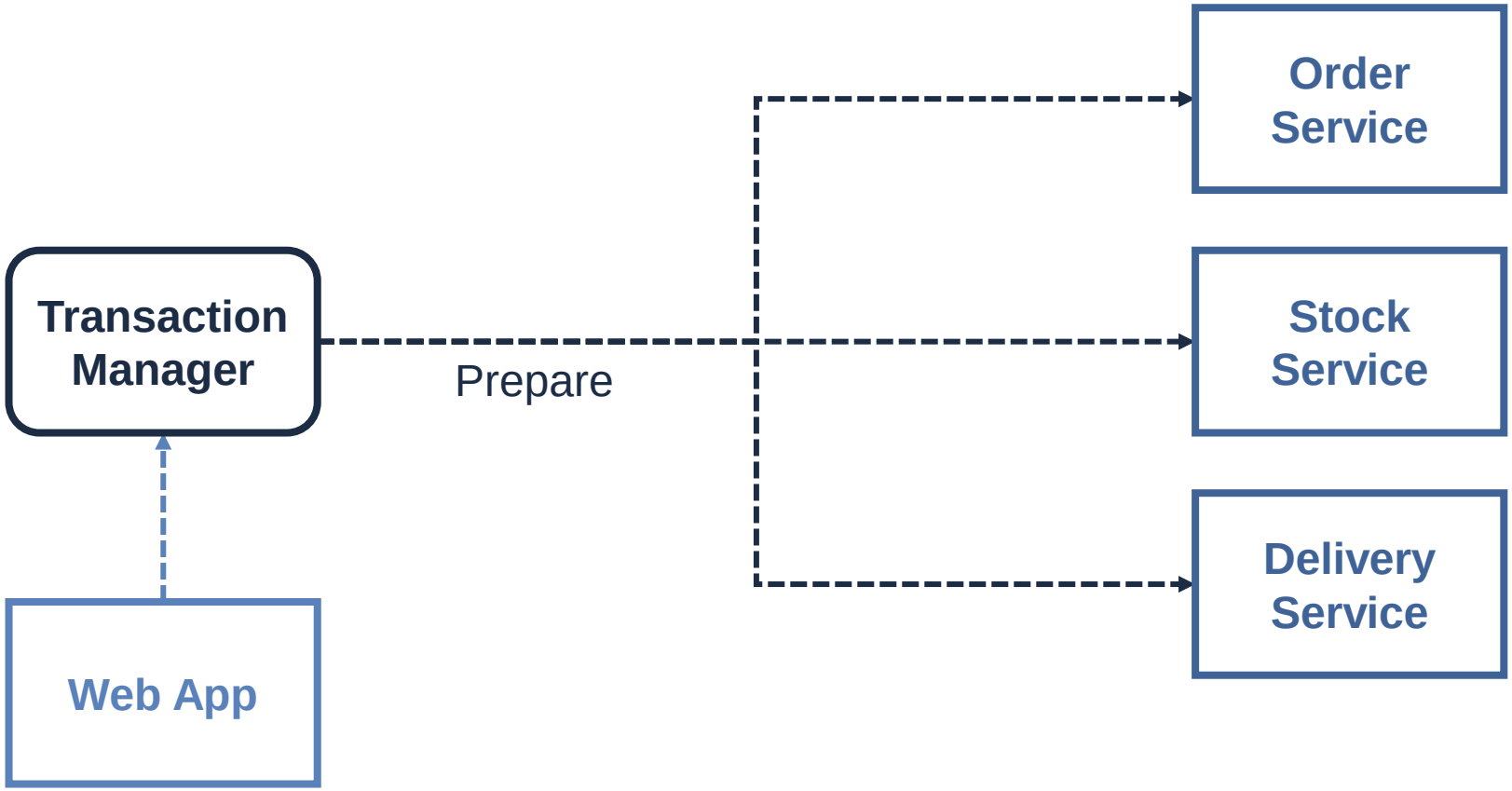
- Transacciones **ACID tradicionales**
  - Atomicidad, consistencia, aislamiento y durabilidad.
- Patrón de confirmación de dos fases **(2PC)**
  - ACID es obligatorio
  - Teorema CAP: elección de consistencia
- Patrón **SAGA**
  - Atomicidad por disponibilidad y consistencia
- Patrón de **consistencia eventual**
  - ACID
  - Teorema CAP: elección de disponibilidad

Confirmación de 2 fases (Two Phase Commit / 2PC)

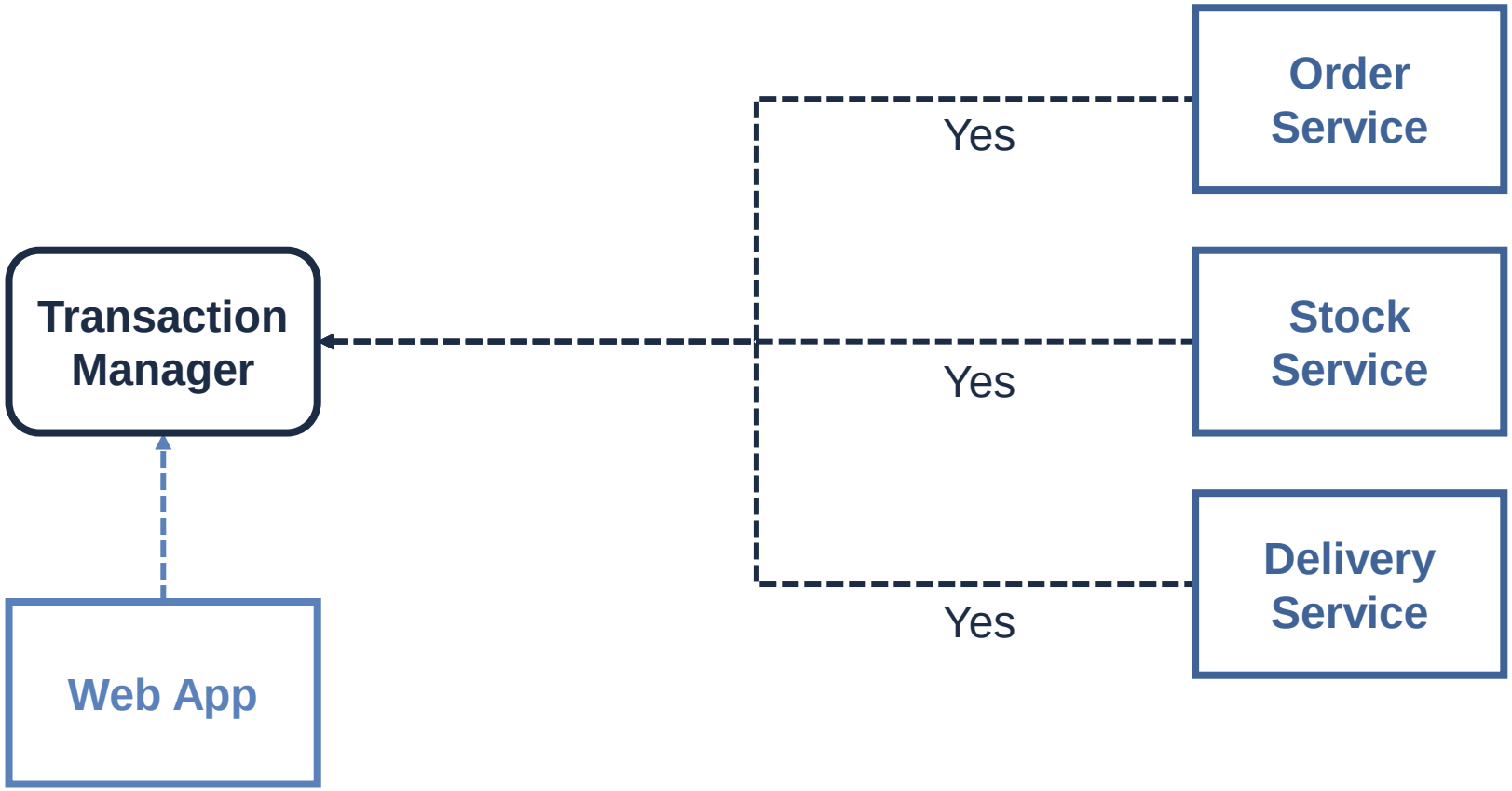


- Patrón para transacciones distribuidas
  - El **administrador de transacciones** maneja las transacciones
- **Fase de preparación**
  - El gestor de transacciones notifica el inicio de preparación.
- **Fase de confirmación**
  - El gestor de transacciones recibe las confirmaciones.
  - Gestor de transacciones
    - Emite un **Commit** en caso todos hayan confirmado.
    - Emite un **Rollback** en caso uno no haya confirmado.

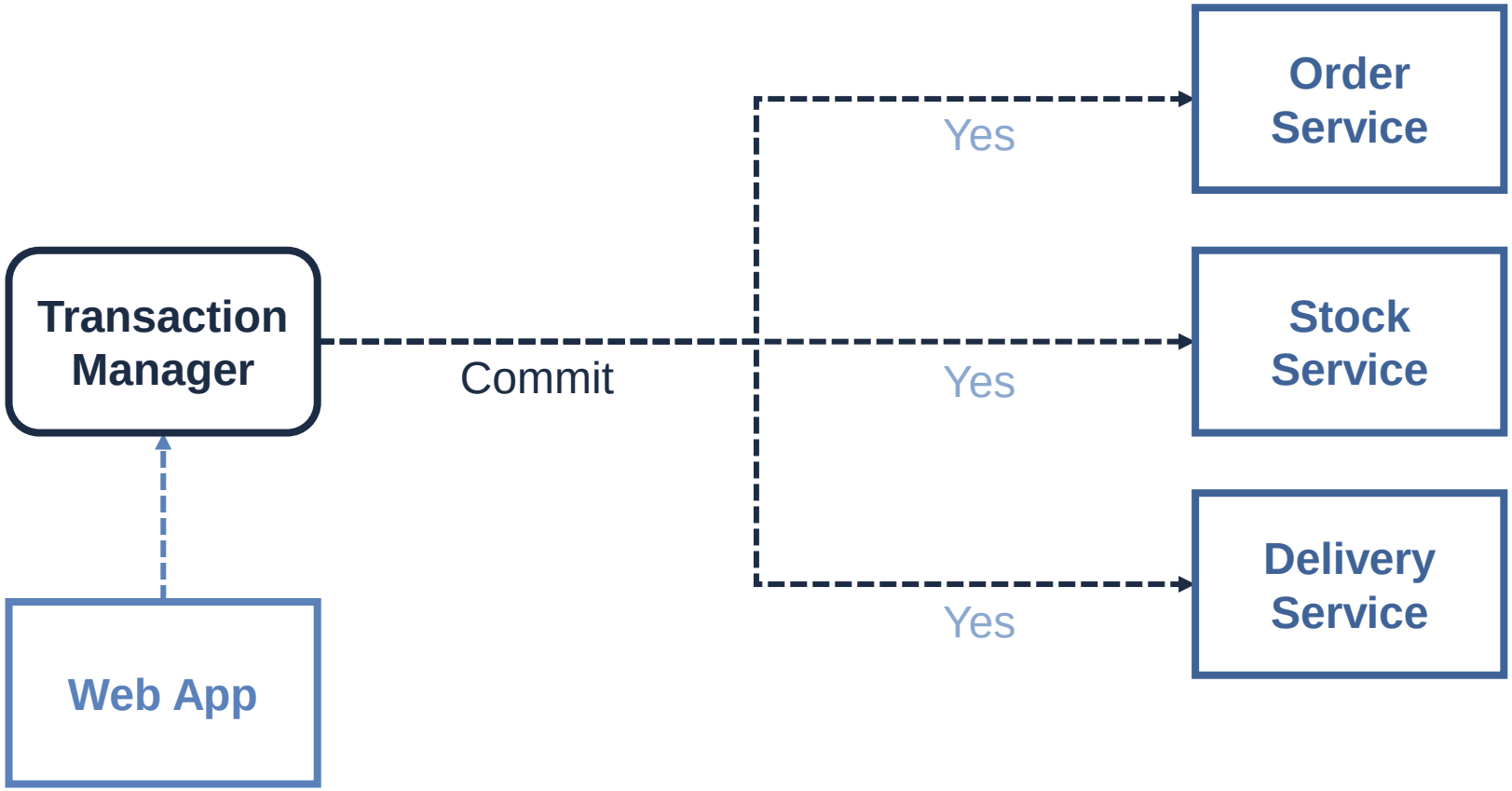
Confirmación de 2 fases – Fase de preparación



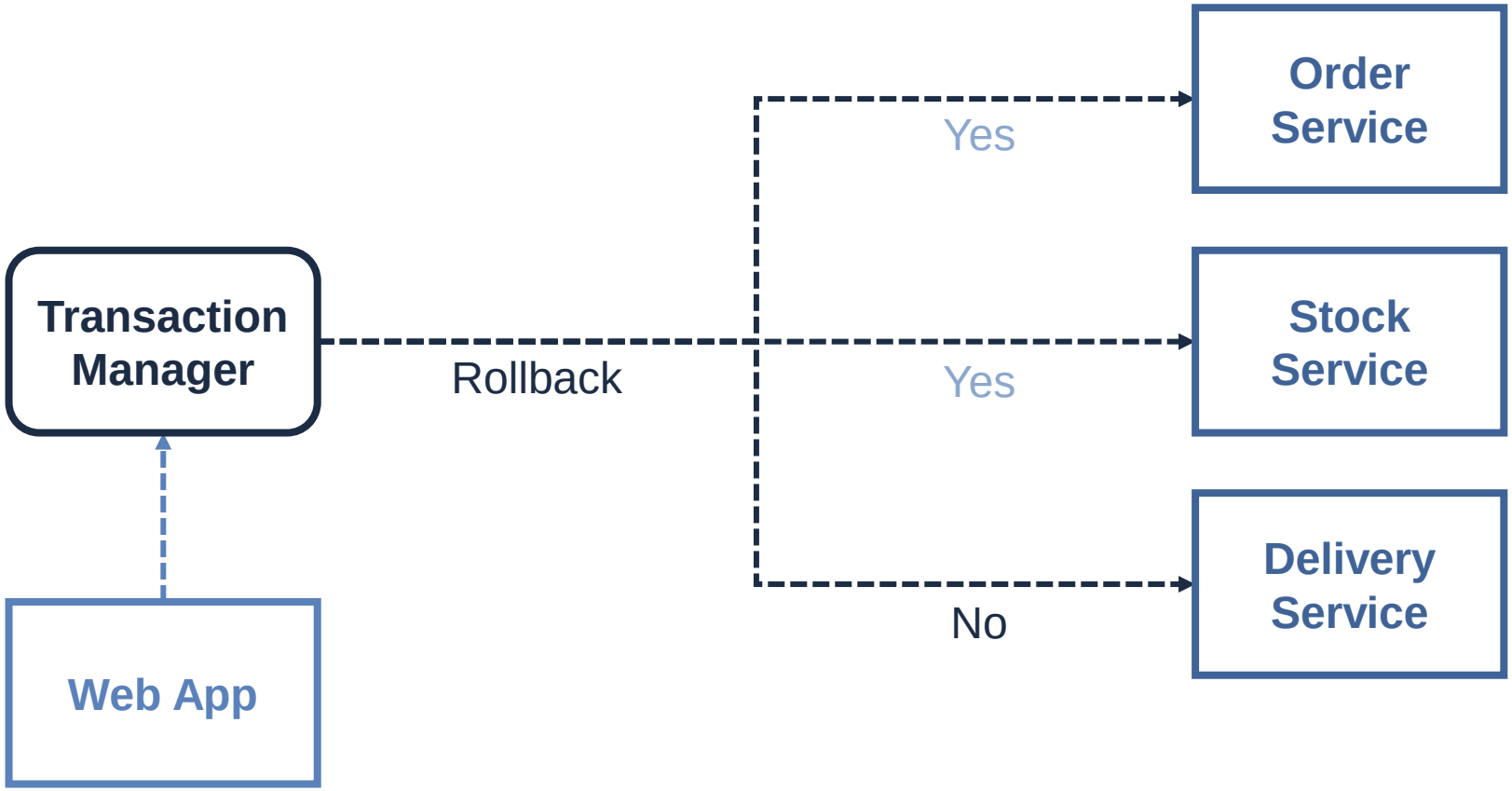
Confirmación de 2 fases – Fase de Confirmación



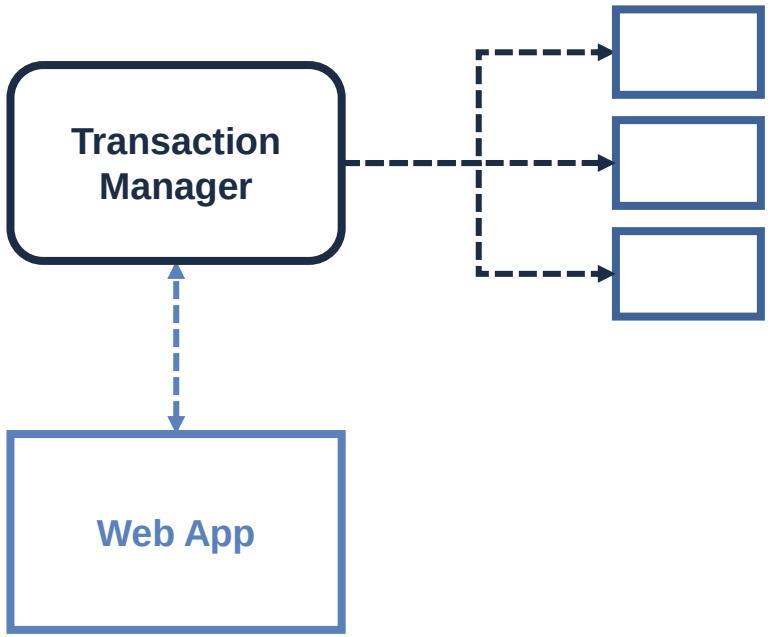
Confirmación de 2 fases – Commit



Confirmación de 2 fases – Rollback



Confirmación de 2 fases (Two Phase Commit / 2PC)



- Advertencias
  - Confianza en un **administrador de transacciones**
  - Sin respuesta de confirmación
  - Fallar después de una confirmación
  - Las transacciones pendientes **bloquean recursos**
  - Evitar implementaciones personalizadas
  - Tiene **problemas de escalado**
  - Rendimiento reducido
  - **ANTIPATRÓN**
- Considerar alternativas
  - Patrón de saga
  - Consistencia eventual

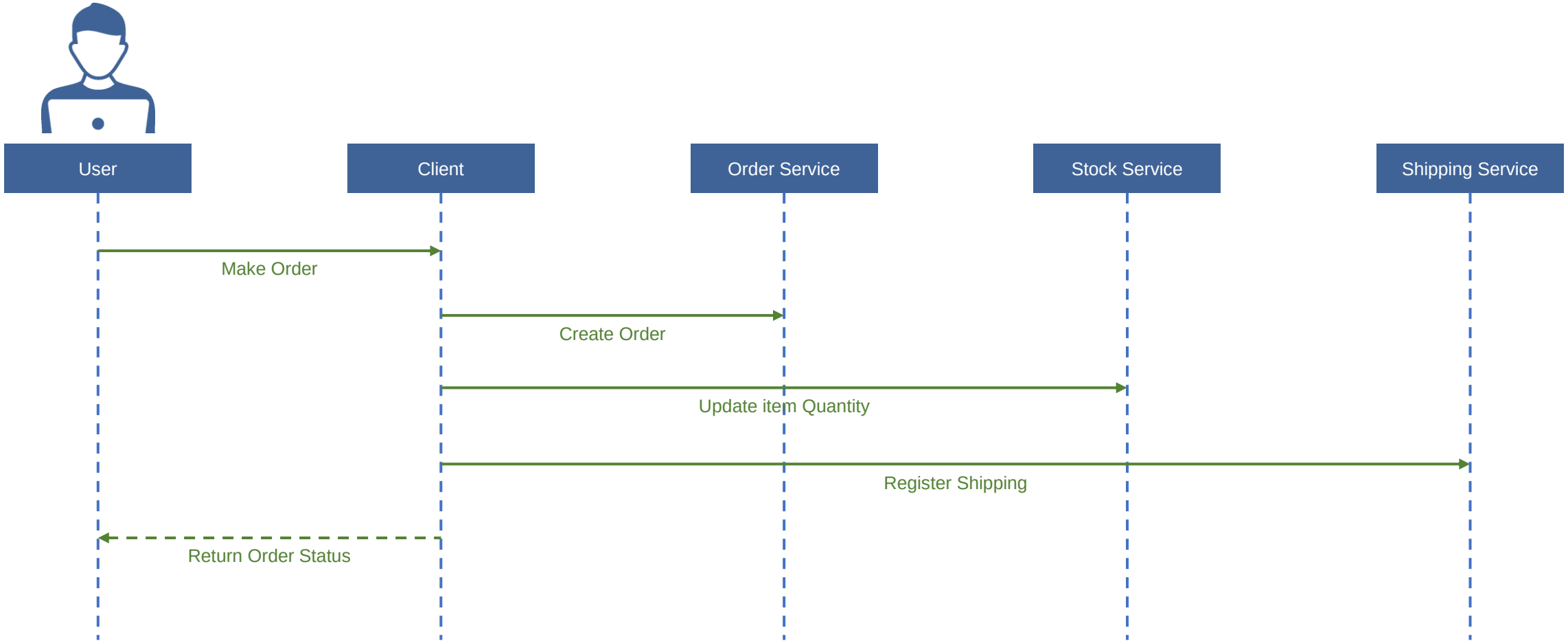
Contexto

Con la adopción del patrón "**Base de datos por servicio**" en la arquitectura de microservicios, significa que cada servicio tiene su propia base de datos.

Existe un problema sobre cómo garantizar la coherencia de los datos entre los servicios.

Por ejemplo, está implementando la función Pedido en un proyecto de Compras en línea. Cuando el usuario final realiza un pedido, la aplicación llamará al Servicio de Stock para actualizar el número de producto en el stock, luego llamará al servicio de envío para entregar el producto solicitado al usuario final.

Contexto


User
Client
Order Service
Stock Service
Shipping Service
Make Order
Create Order
Update item Quantity
Register Shipping
Return Order Status

¿Cuál es el problema?

- ¿Qué sucede si la cantidad del producto es menor que la cantidad que pide el usuario final?
- ¿O qué sucede si hay un problema con el servicio de envío (shipping) y la aplicación no puede registrar el envío correctamente?

Los pedidos, el artículo y el envío están en diferentes bases de datos, por lo que no podemos usar una transacción ACID local. Esa es la razón por la que no se puede usar uno de los tipos de transacciones que se llama "Confirmación de dos fases" - "2PC".

Las transacciones distribuidas, como el protocolo de confirmación en dos fases (2PC), requieren que todos los participantes de una transacción la confirmen o reviertan antes de que la transacción pueda continuar. **No obstante, algunas implementaciones de participantes, como las bases de datos NoSQL y la administración de mensajes, no admiten este modelo.**

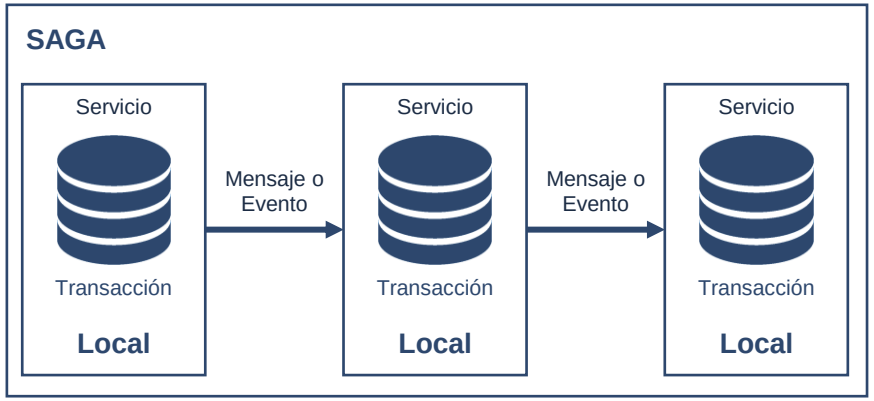
SAGA

El patrón de saga proporciona **la administración de transacciones mediante una secuencia de transacciones locales.**

Una transacción local es el esfuerzo del trabajo atómico realizado por un participante de la saga.

Cada transacción local actualiza la base de datos y publica un mensaje o evento para desencadenar la siguiente transacción local en la saga.

Si se produce un error en una transacción local, la saga ejecuta una serie de transacciones de compensación que deshacen los cambios realizados por las transacciones locales anteriores.



SAGA

- **Las transacciones compensables** son transacciones que se pueden invertir procesando otra transacción con el efecto opuesto.
- Una **transacción dinámica** es el punto en el que se debe continuar o no continuar en una saga. Si se confirma la transacción dinámica, la saga se ejecuta hasta su finalización. Una transacción dinámica puede ser una transacción que no es compensable ni reintentable, o puede ser la última transacción compensable o la primera transacción reintentable de la saga.
- **Las transacciones reintentables** son transacciones que siguen a la transacción dinámica y que está garantizado que se realizarán correctamente.

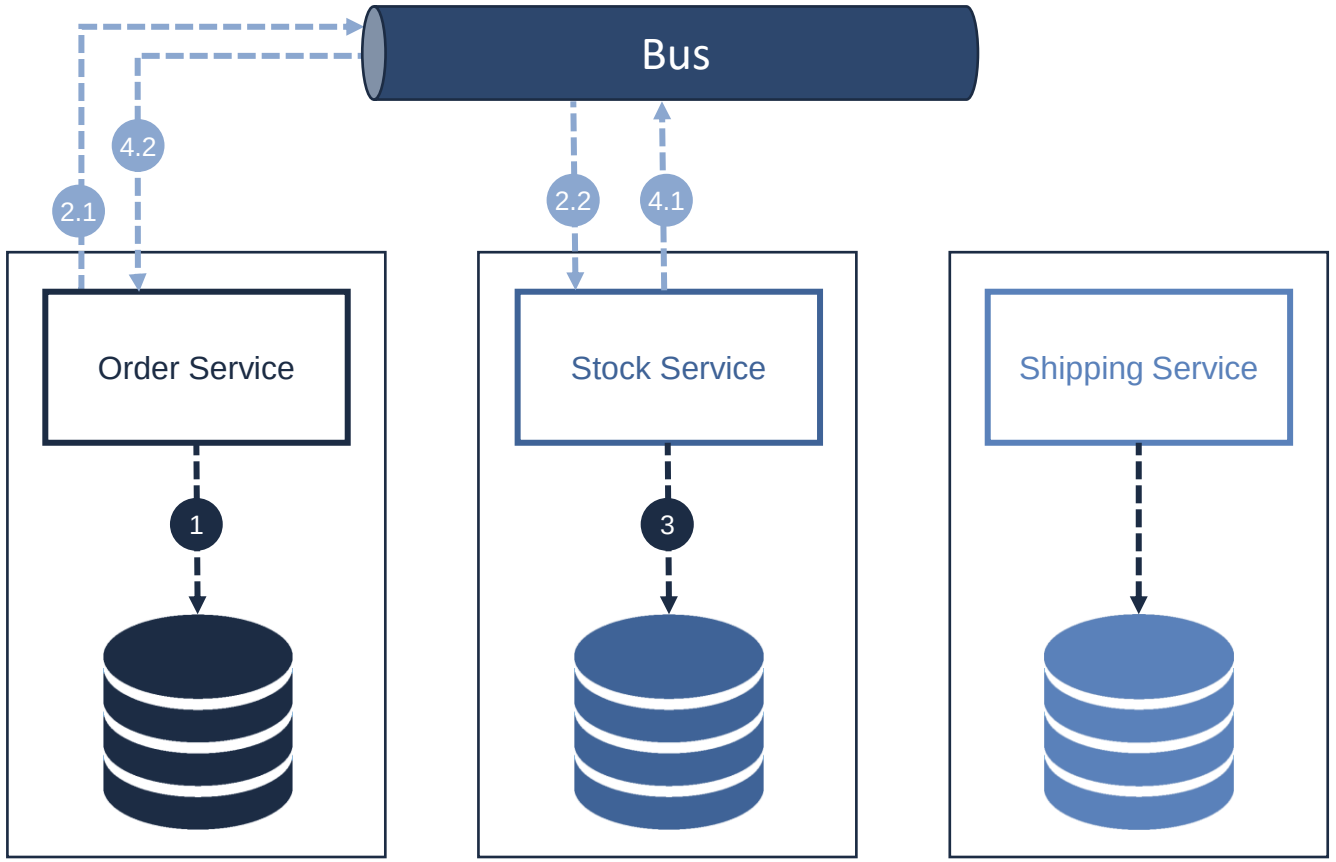
02

Patrón SAGA Choreography

SAGA - Corografía

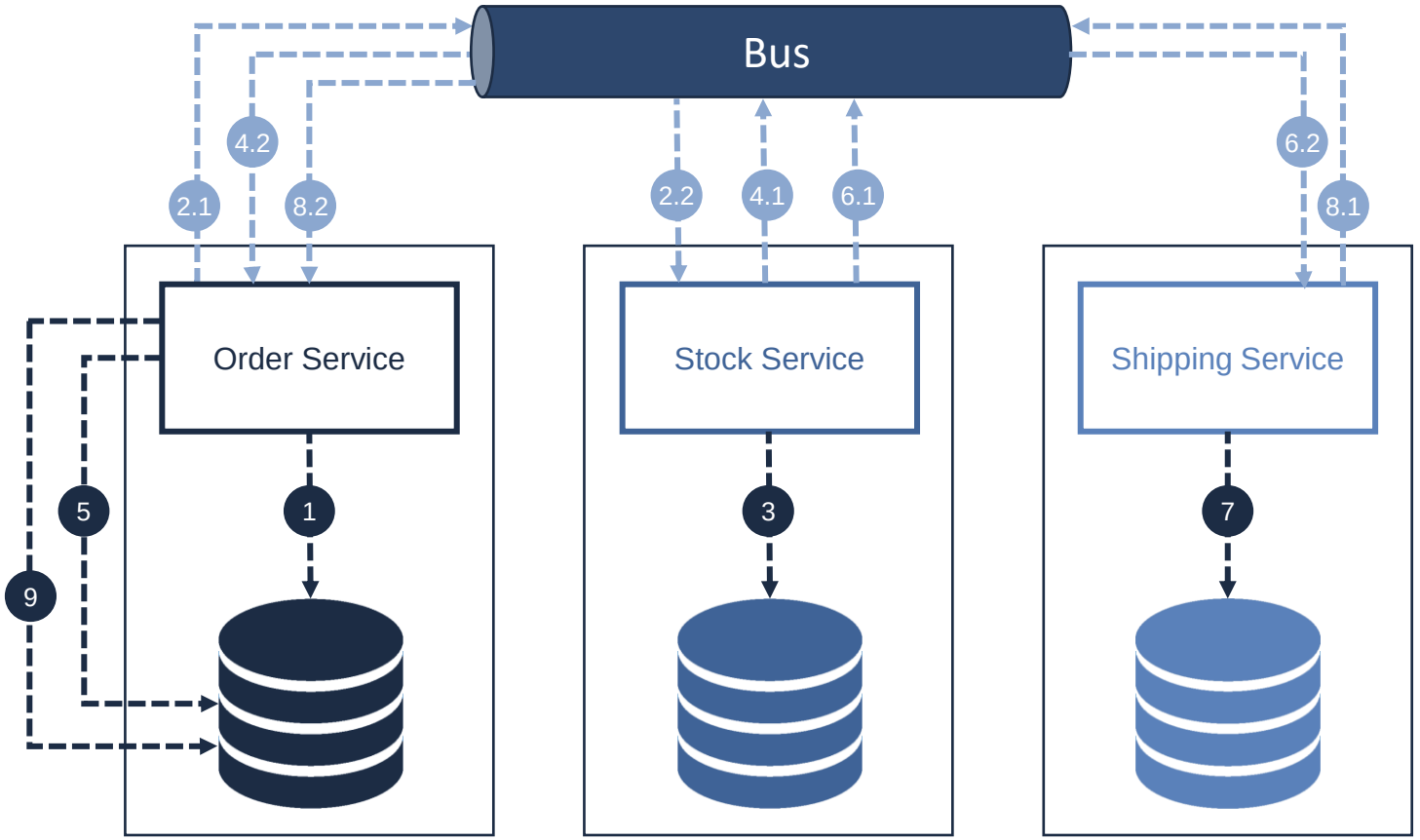
La coreografía es una **secuencia de pasos o movimientos en la danza**... y también es similar en el desarrollo de software si aplicas SAGA - Coreografía Con este enfoque, **cada servicio produce y escucha los eventos de otros servicios**, en función del evento de otros servicios decidirá la siguiente acción.

Flujo de SAGA-Choreography



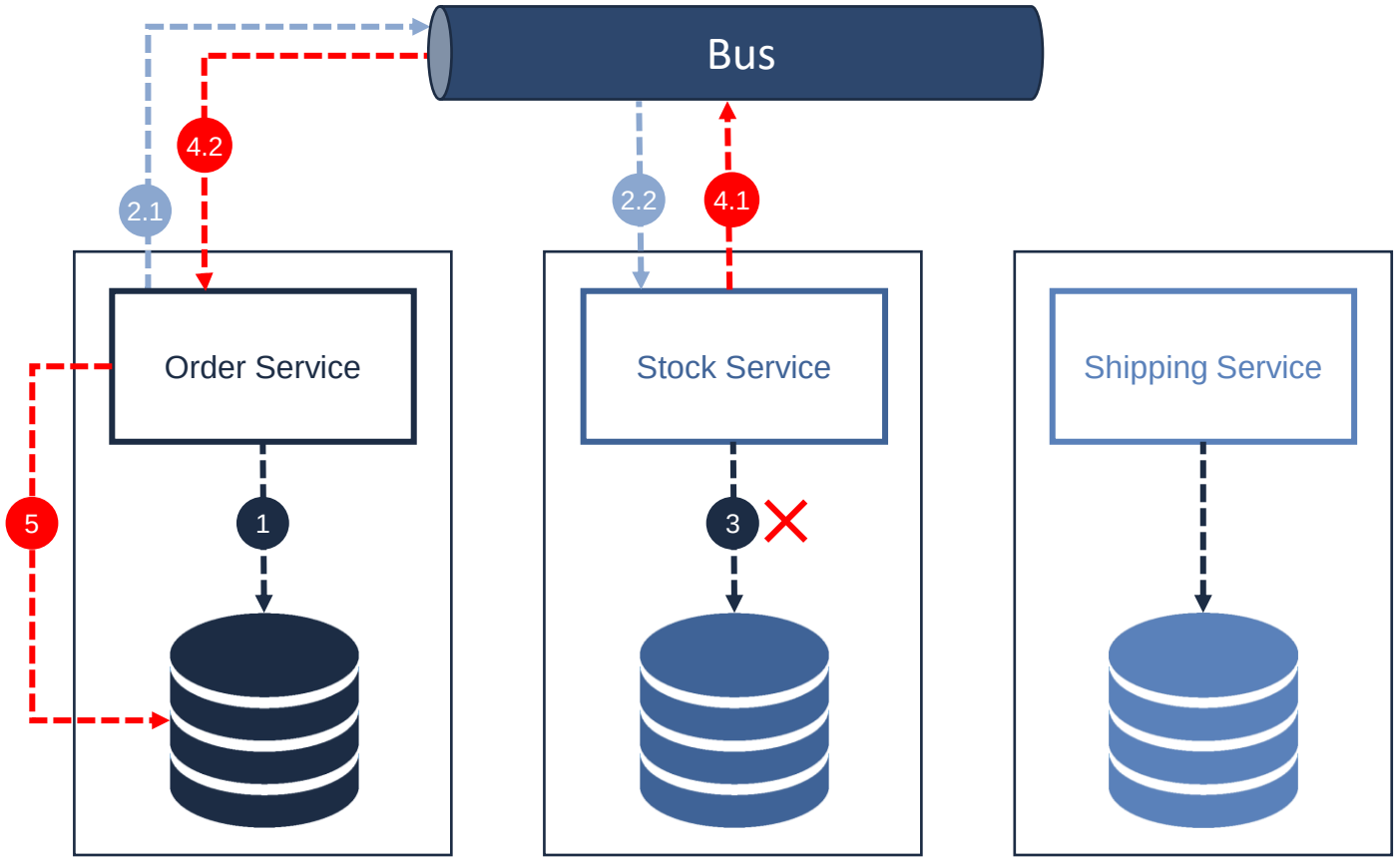
1. El servicio de pedidos crea un registro en la base de datos con el estado "**Verificando**".
2. El servicio de pedidos publica un evento "**El pedido se ha creado correctamente**" y el servicio de Stock escuchará este evento.
3. El servicio de stock actualiza el número de productos de la base de datos de stock.
4. El servicio de stock publica un evento "**El producto se ha actualizado correctamente**" y el **servicio de pedidos** y el servicio de envío escucharán este evento.

Flujo de SAGA-Choreography



5. El servicio de pedido cambiará el estado del pedido a "**Envío**".
6. El servicio de stock publica un evento "**El producto se ha actualizado correctamente**" y el servicio de pedidos y el **servicio de envío** escucharán este evento.
7. El servicio de envío crea un registro en la base de datos de envíos.
8. El servicio de envío publica un evento "**El envío se ha registrado correctamente**" y el servicio de pedidos escuchará este evento.
9. El Servicio de Pedido cambia el estado del Pedido a "**Finalizar**".

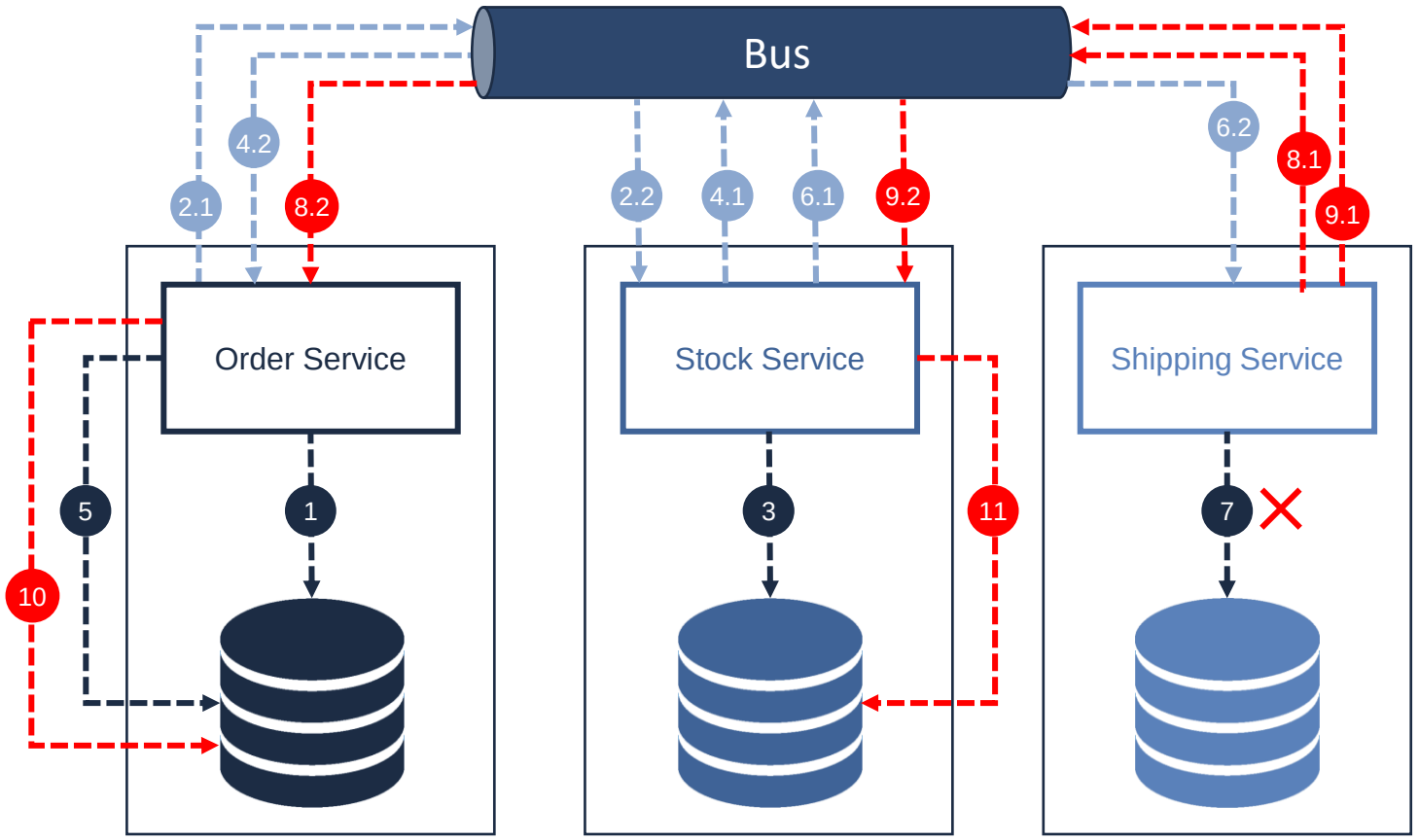
Flujo de Rollback en SAGA-Choreography (Escenario 1)



Hay un error con Stock Service.

1. El servicio de pedidos crea un registro en la base de datos con el estado "**Verificando**".
2. El servicio de pedidos publica un evento "**El pedido se ha creado correctamente**" y el servicio de Stock escuchará este evento.
3. El servicio de stock actualiza el número de productos de la base de datos de stock. **Pero hay un error inesperado al guardar el registro del producto.**
4. Stock Service publica un evento de "**reversión**" "**Error al actualizar el número de productos en stock**".
5. El servicio de pedido cambiará el estado del pedido a "**Error**".

Flujo de Rollback en SAGA-Choreography (Escenario 2)



El servicio de pedidos y el servicio de stock se ejecutan correctamente. Hay un error con el servicio de envío.

7. El servicio de envío crea un registro en la base de datos de envíos. **Pero hay un error inesperado al guardar**.

8. El servicio de envío publica un evento de "**reversión**" "**Error al registrar el servicio de envío**" y tanto el servicio de pedidos.

9. El servicio de envío publica un evento de "**reversión**" "**Error al registrar el servicio de envío**" y tanto el servicio de stock.

10. El servicio de pedido cambiará el estado del pedido a "**Error**".

11. El servicio de stock actualiza el número del producto.

Flujo de SAGA-Choreography

Tenga en cuenta que es crucial **definir un ID compartido común para cada transacción**, de modo que cada vez que lance un evento, todos los oyentes puedan saber de inmediato a qué transacción se refiere.

Ventajas y desventajas de SAGA - Coreografía

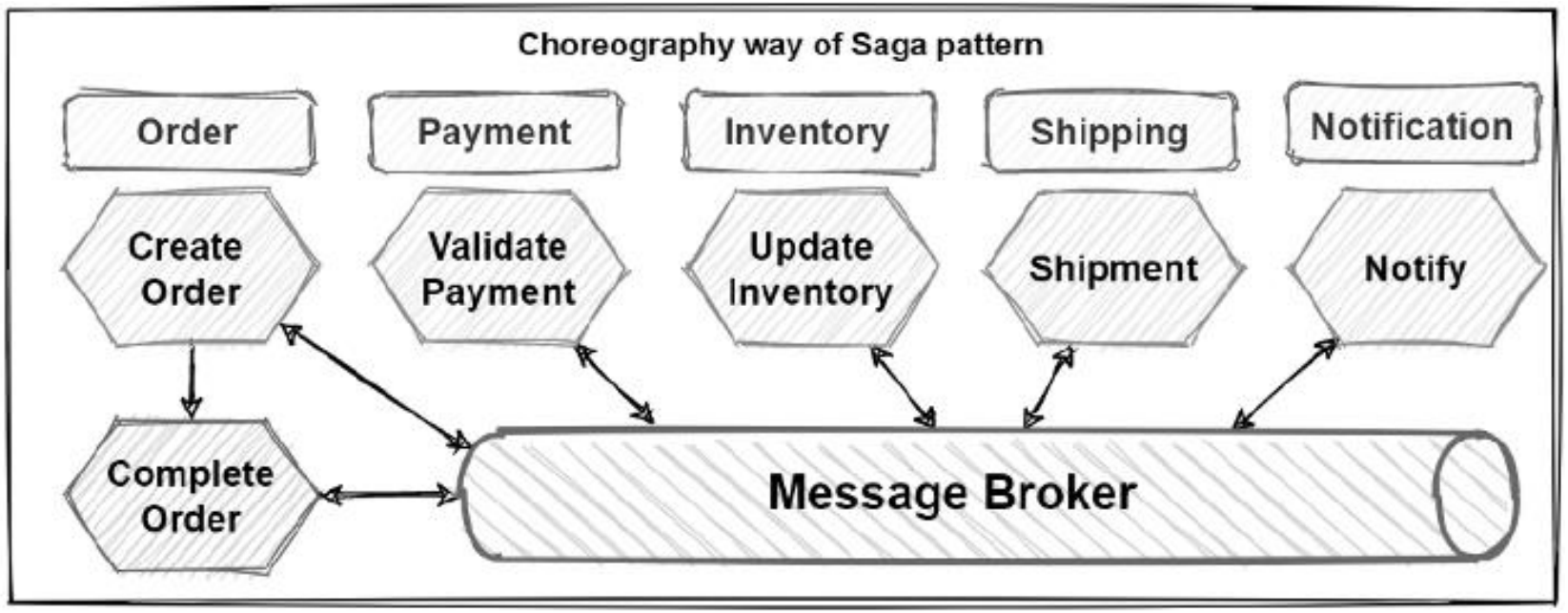
- La **coreografía** es la forma más sencilla de implementar el patrón SAGA. Es muy fácil de entender y no requiere demasiado esfuerzo para construir. **Cada transacción local en cadena es independiente** ya que no tienen conocimiento directo entre sí. **Si su transacción distribuida solo incluye de 2 a 4 transacciones locales, entonces la coreografía es una opción muy adecuada.**

- Tener **demasiadas transacciones locales** hará que el seguimiento de qué servicios escuchan qué eventos, **se vuelva muy complejo**. Con la muestra anterior, solo tenemos 3 transacciones locales pero más de 10 pasos para manejar. Imaginemos que cuando tengamos 10 transacciones locales, **¿qué sucederá?**

DEMO

GALAXY TRAINING

SAGA Choreography



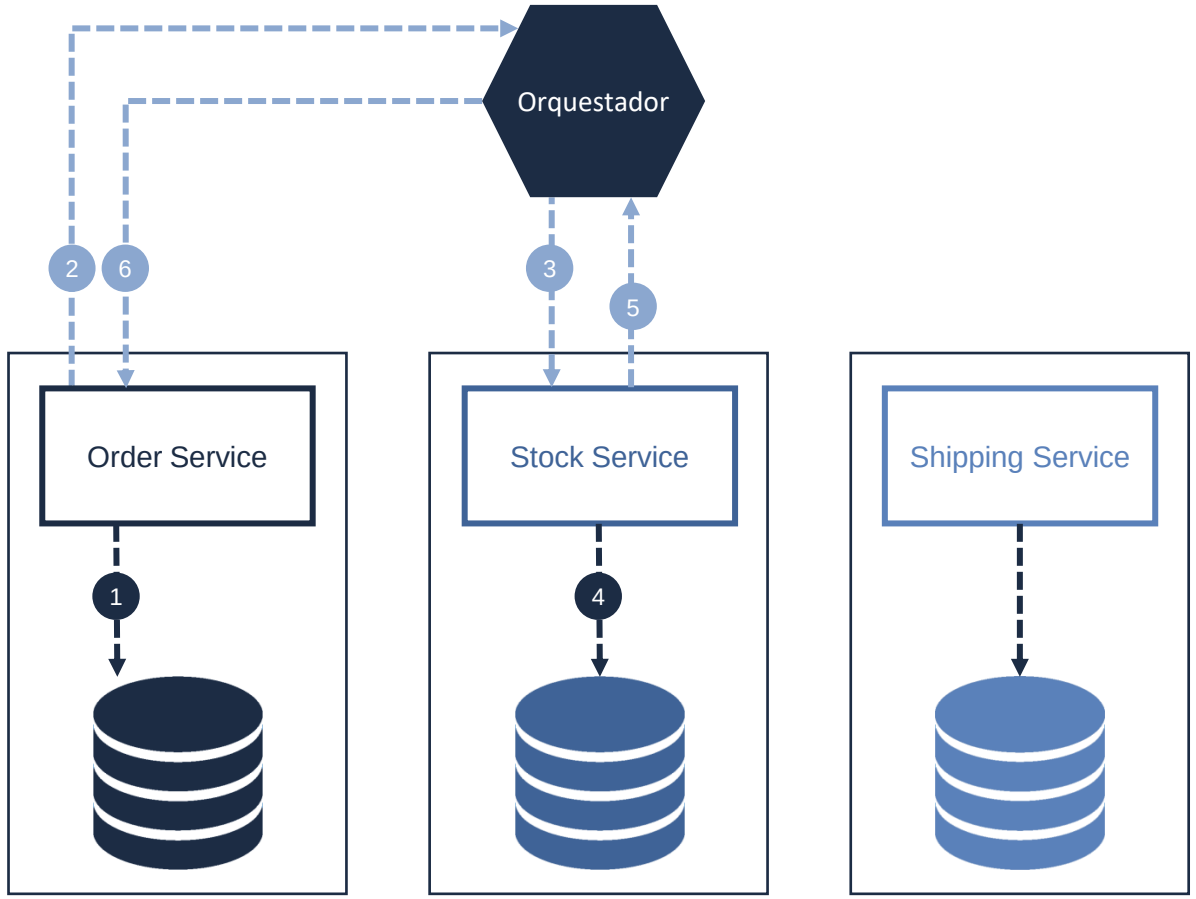
03

Patrón SAGA Orchestration.

SAGA - Orquestación

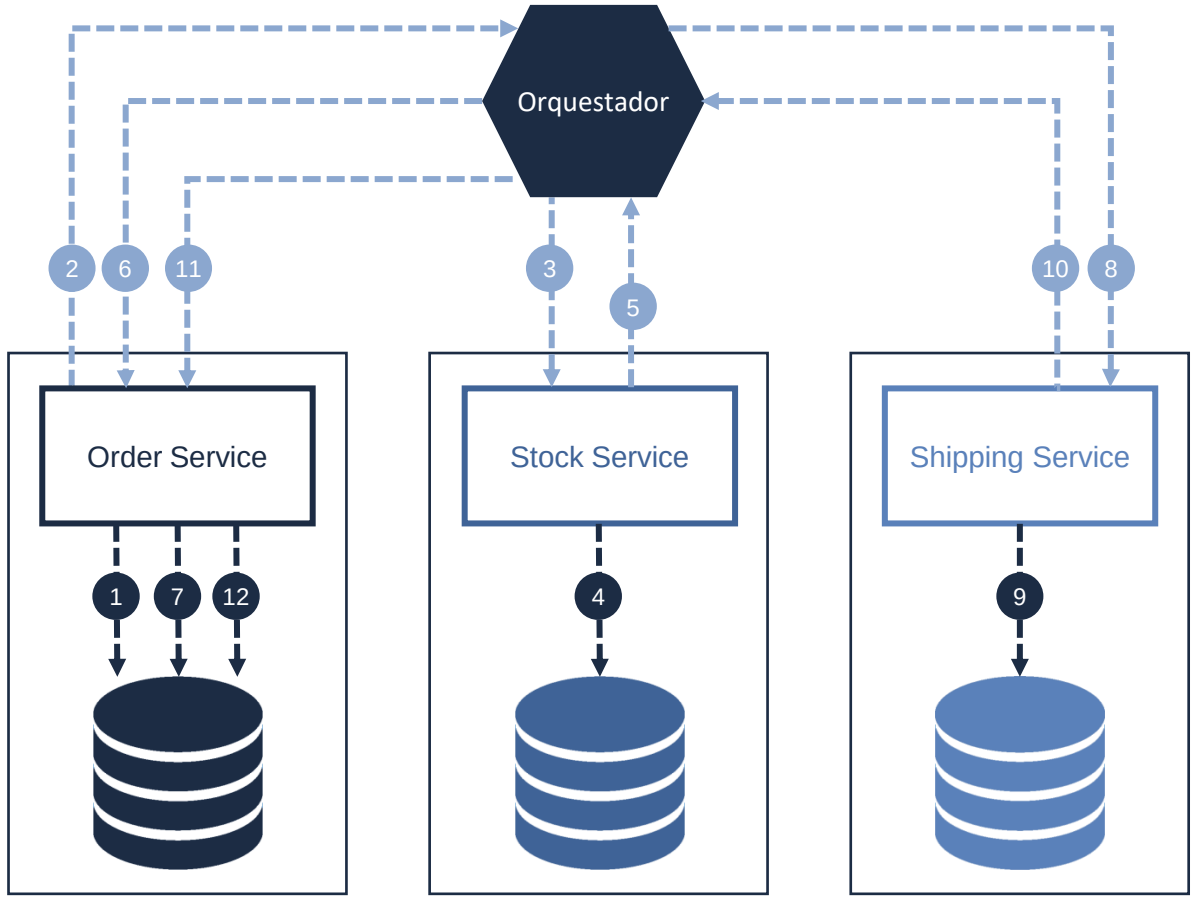
Para abordar la complejidad del patrón SAGA, es bastante normal agregar un **administrador de procesos como orquestador**. El administrador de procesos es responsable de escuchar eventos y activar puntos finales. Volviendo a la muestra anterior, en lugar de que el servicio de pedido, el servicio de stock y el servicio de envío tengan que escucharse entre sí, implementaremos una **"gestión de procesos"** y estos 3 servicios escucharán esto.

Flujo de SAGA-Orchestration



1. El servicio de pedidos crea un registro de pedido en la base de datos con el estado "**Verificando**".
2. El servicio de pedido Solicita al Orquestador iniciar la "**Transacción de pedido**".
3. El Orquestador envía un "**Comando de actualización de stock**" al servicio de stock.
4. El servicio de stock actualiza el número del producto solicitado en la base de datos.
5. El servicio de stock responde al Orquestador con el mensaje "**Stock actualizado correctamente**".
6. El Orquestador envía un "**Comando de actualización de pedidos**" al servicio de pedidos con información de que el stock ya se actualizó correctamente.

Flujo de SAGA-Orchestration



7. Actualización del servicio de pedidos. Registro de pedidos en la base de datos con el estado "**Envío**".

8. El Orquestador envía un "**Comando de envío de registro**" al servicio de envío.

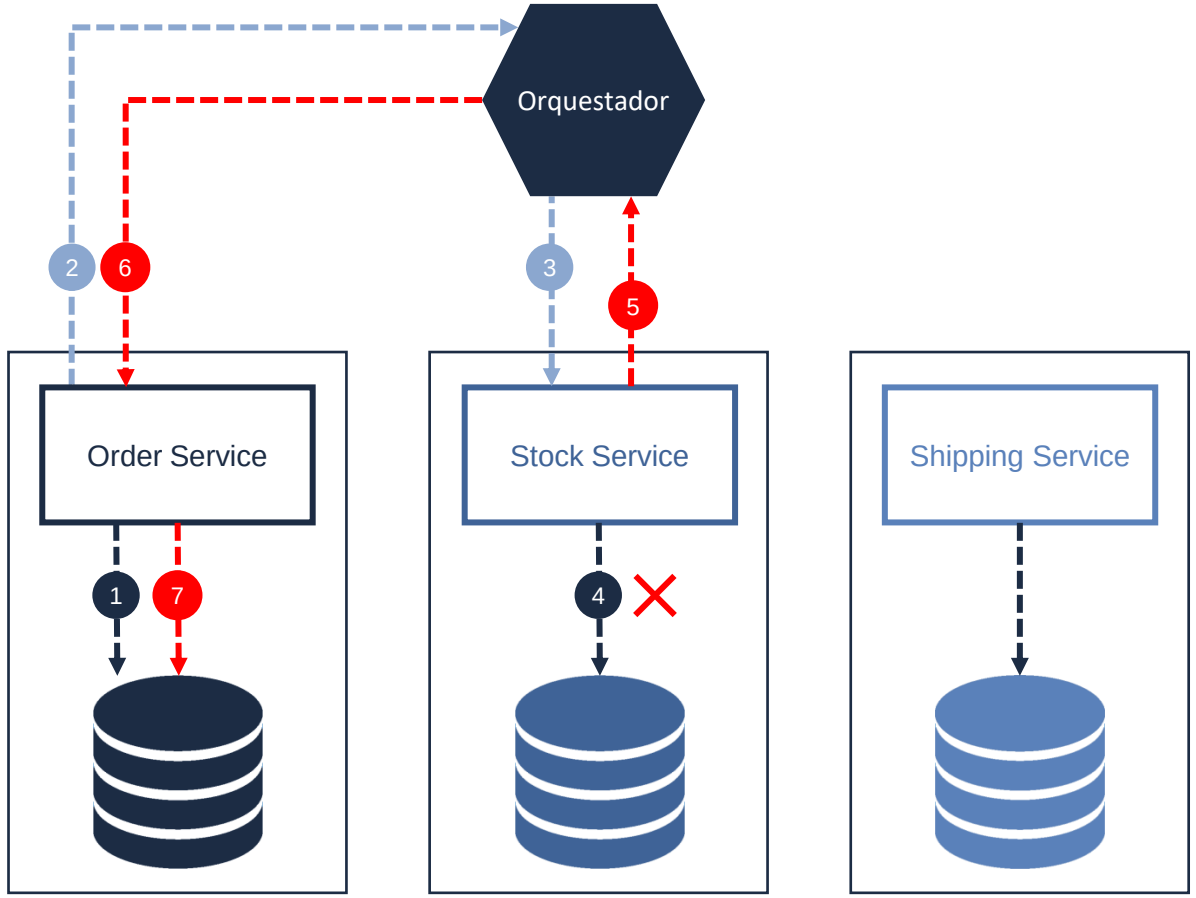
9. El servicio de envío inserta un nuevo registro en la base de datos de envíos.

10. El servicio de envío responde al Orquestador con el mensaje "**El envío se ha registrado correctamente**".

11. El Orquestador envía un "**Comando de actualización de pedido**" al servicio de pedidos con información de que el envío ya se actualizó correctamente.

12. El servicio de pedidos actualiza el registro de pedidos en la base de datos con el estado "Finalizar".

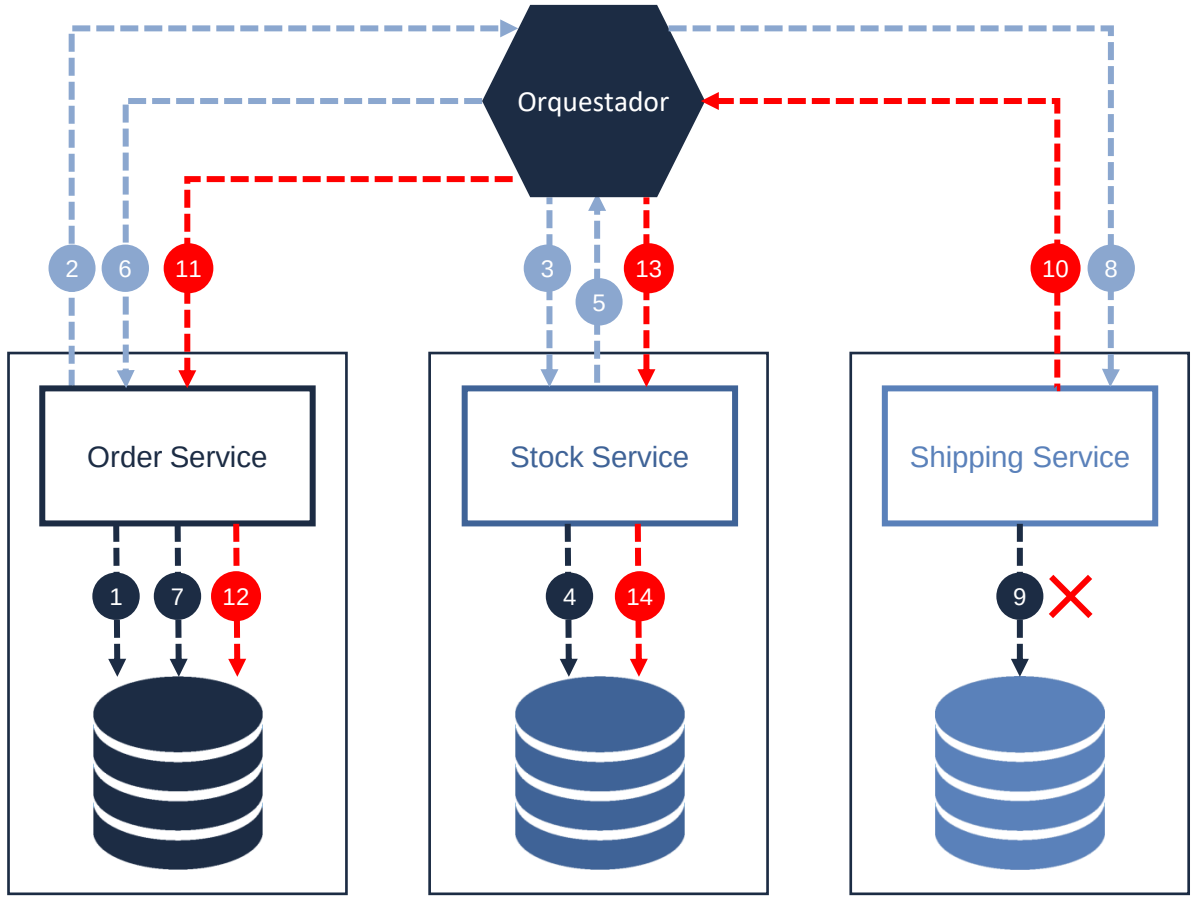
Flujo de Rollback en SAGA-Orchestration (Escenario 1)



Hay un error con el servicio Stock

1. El servicio de pedidos crea un registro de pedido en la base de datos con el estado "**Verificando**".
2. El servicio de pedido Solicita al Orquestador iniciar la "**Transacción de pedido**".
3. El Orquestador envía un "**Comando de actualización de stock**" al servicio de stock.
4. El servicio de stock actualiza el número del producto solicitado en la base de datos. **Pero hay un error.**
5. El servicio de stock responde con el mensaje "**Agotado**" al Orquestador.
6. El Orquestador envía un **"Comando de orden de reversión**" al servicio de pedidos con información de que Stock fallo en la actualización.
7. El servicio de pedidos actualiza el registro de pedidos en la base de datos con el estado "**Error**".

Flujo de Rollback en SAGA-Orchestration (Escenario 2)



Hay un error con el servicio de envío

10. El servicio de envío responde con el mensaje "**Error al registrar envío**" al Orquestador.

11. El Orquestador envía un "**Comando de pedido de reversión**" al servicio de pedidos con información de que el servicio de envío fallo en el registro.

12. El servicio de pedidos actualiza el registro de pedidos en la base de datos con el estado "**Error**".

13. El Orquestador envía un "**Comando de reversión de stock**" al servicio de stock con información de que el servicio de envío fallo en el registro.

14. El servicio de stock actualiza el número del producto.

Flujo de SAGA-Orchestration

Una forma estándar de modelar una SAGA-Orchestrator es **una máquina de estado** donde cada transformación corresponde a un comando o mensaje. Las máquinas de estado son un excelente patrón para estructurar un comportamiento bien definido ya que son fáciles de implementar.

Puede utilizar servicios en la nube que pueden ayudarle, por ejemplo:

https://theburningmonk.com/2017/07/applying-the-saga-pattern-with-aws-lambda-and-step-functions/
https://www.freecodecamp.org/news/an-introduction-to-azure-durable-functions-patterns-and-best-practices-b1939ae6c717/

Ventajas y desventajas de SAGA - Orchestration

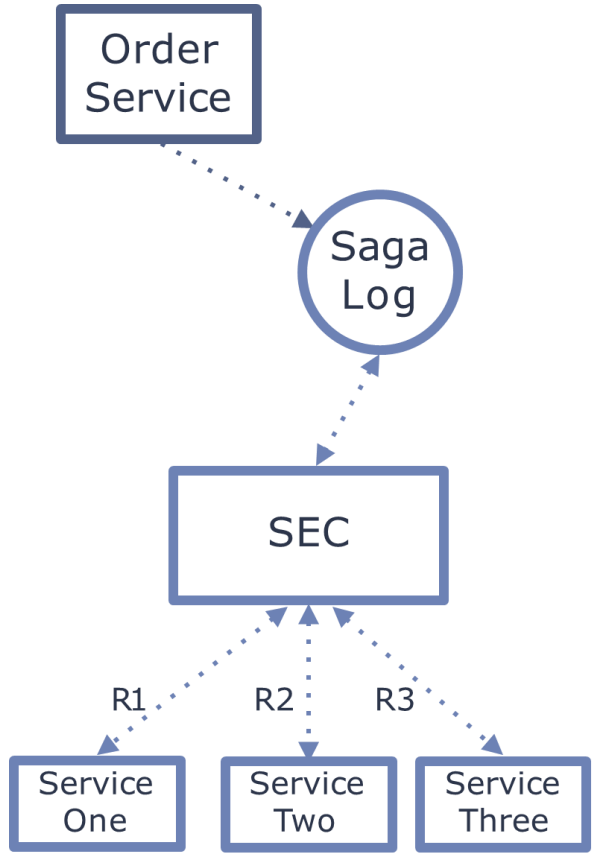
- **Fácil de mantener todo el flujo de trabajo** en un solo lugar: **Orquestador**. **Evita las dependencias cíclicas** entre servicios. Todos los servicios solo se comunican con el Orquestador. La complejidad de la transacción sigue siendo lineal cuando se agregan nuevos pasos.

- Sin embargo, para lograr las ventajas anteriores, debe **diseñar un orquestador inteligente**. Además, tener **demasiada lógica en el orquestador** podría llevar a dificultades en el mantenimiento.

04

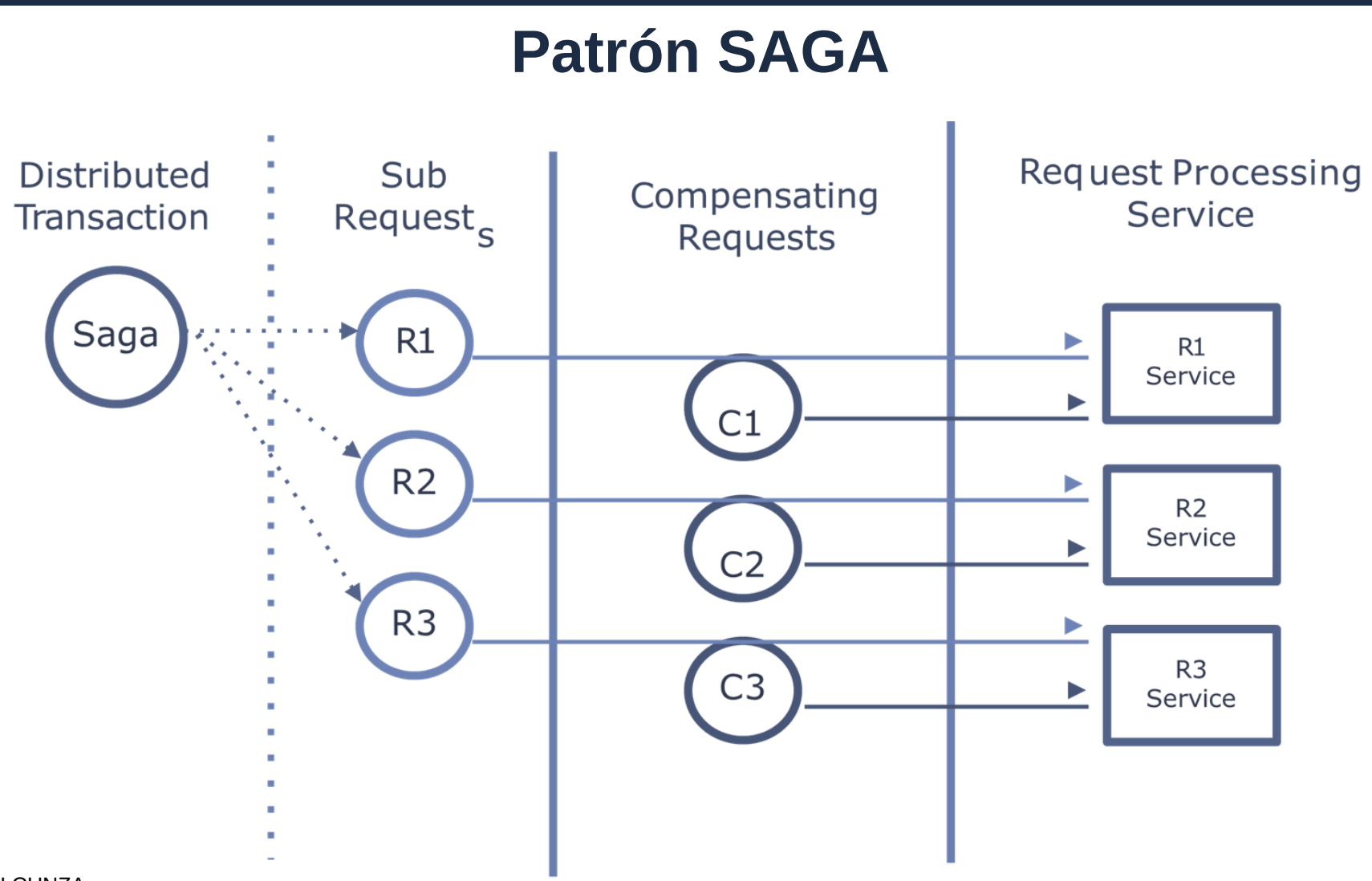
Implementación genérica del patrón SAGA a un proceso.

Patrón SAGA



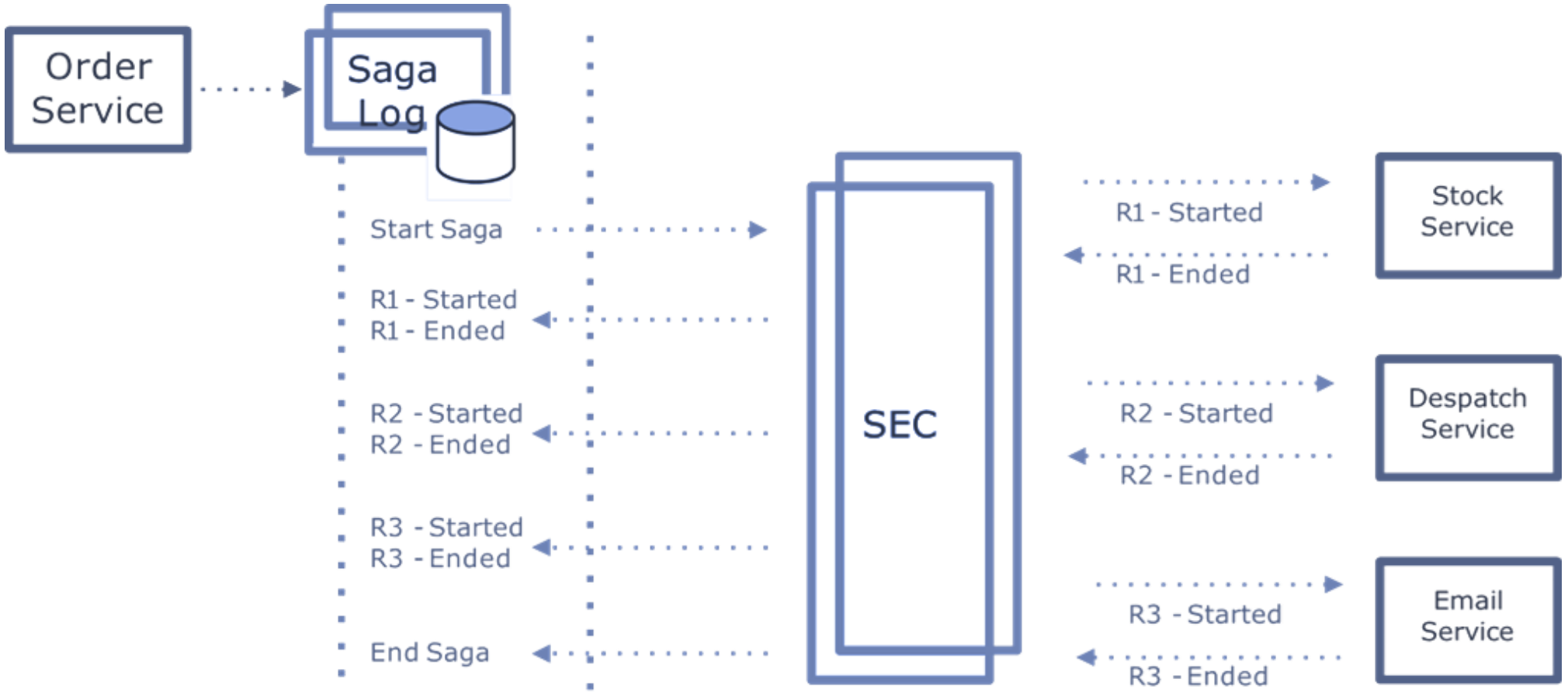
- **Reemplaza una transacción distribuida con una saga**
  - Divide la transacción en muchas solicitudes
  - Rastrea cada solicitud
  - ACID: compromiso de atomicidad
  - Descrito por primera vez en 1987
- También es un patrón de **gestión de fallas**
  - Qué hacer cuando falla un servicio
  - **Compensar solicitudes**
- Implementación
  - **SAGA Log**
  - Coordinador de ejecución de saga (**SEC**)
  - Solicitudes y compensación de solicitudes

Implementación genérica del patrón SAGA



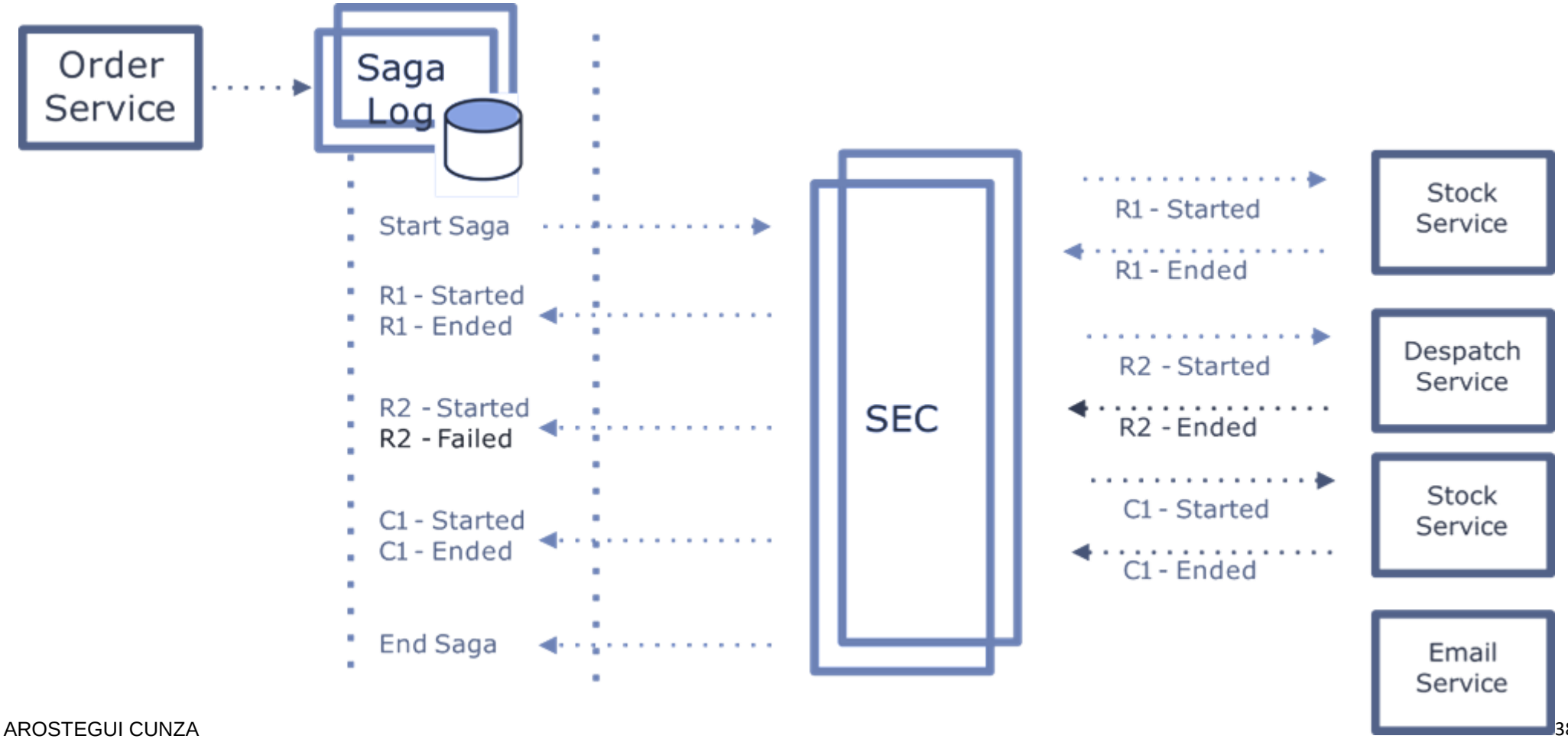
Implementación genérica del patrón SAGA

Saga exitoso

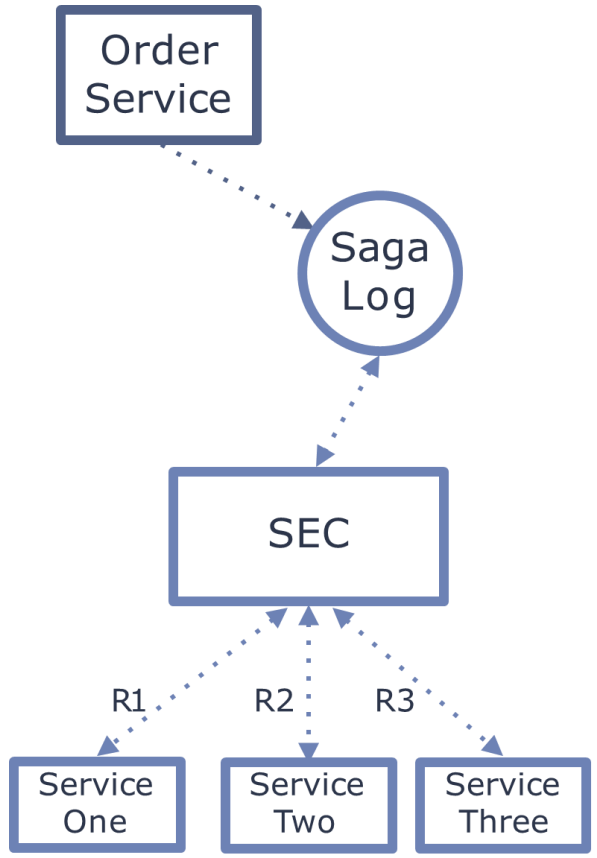


Implementación genérica del patrón SAGA

Saga NO exitoso



Patrón SAG implementación

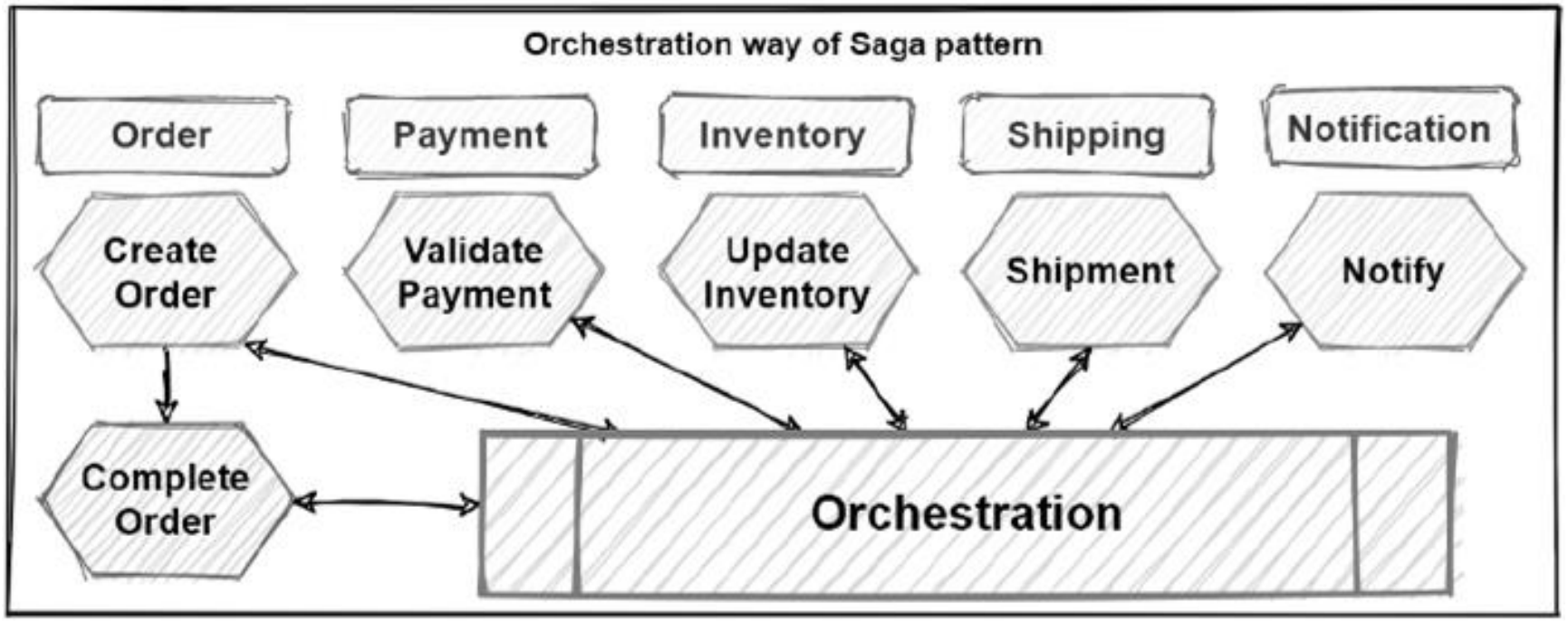


- Servicio que inicia la saga.
  - Envía la solicitud de saga al registro de saga
- **Saga Log**
  - Servicio con una base de datos
- **SEC**
  - Interpreta y escribe en el registro.
  - Envía solicitudes de saga
  - Envía solicitudes de compensación de saga
  - Recuperación: estado seguro vs estado inseguro
- **Solicitudes de Compensación**
  - Enviar en caso de error para todas las solicitudes completadas
  - **Idempotente (fácil con REST)**
  - Cada uno se envía de cero a muchas veces

DEMO

GALAXY TRAINING

SAGA Orchestration



05

GALAXY
TRAINING

Recomendaciones para su implementación.

Recomendaciones para su implementación

Tenga en cuenta los puntos siguientes al implementar el patrón de saga:

- El patrón de saga inicialmente **puede ser desafiante**, ya que requiere una nueva forma de pensar sobre cómo coordinar una transacción y **mantener la coherencia de los datos** para un proceso empresarial que abarque varios microservicios.

- El patrón de saga es especialmente **difícil de depurar** y **la complejidad crece a medida que los participantes aumentan**.

- **NO SE PUEDEN REVERTIR LOS DATOS** porque los participantes de la saga confirman los cambios en sus bases de datos locales.

Recomendaciones para su implementación

Tenga en cuenta los puntos siguientes al implementar el patrón de saga:

- La implementación debe ser capaz de **controlar un conjunto de posibles errores** transitorios y proporcionar idempotencia para reducir los efectos secundarios y **garantizar la coherencia de los datos**. La idempotencia significa que la misma operación se puede repetir varias veces sin cambiar el resultado inicial.

- Es mejor **implementar la observabilidad para supervisar y realizar un seguimiento** del flujo de trabajo de la saga.

- La falta de aislamiento de los datos de los participantes **impone desafíos de durabilidad**. La implementación de la saga debe **incluir contramedidas** para reducir las anomalías.

Recomendaciones para su implementación

Pueden producirse las siguientes **anomalías si no se toman las medidas adecuadas**:

- **Pérdida de actualizaciones**, cuando un patrón de saga escribe sin leer los cambios realizados por otro patrón de saga.
- **Lecturas de datos sucios**, cuando una transacción o una saga lee las actualizaciones realizadas por una saga que todavía no ha completado dichas actualizaciones.
- **Lecturas aproximadas/no repetibles**, cuando diferentes pasos de la saga leen datos diferentes porque se produce una actualización de datos entre las lecturas.

Recomendaciones para su implementación

Entre las contramedidas sugeridas para reducir o evitar anomalías se incluyen:

- **El bloqueo semántico**, un bloqueo de nivel de aplicación en el que una transacción de compensable de una saga utiliza un semáforo para indicar que hay una actualización en curso.

- **Las actualizaciones conmutativas** que se pueden ejecutar en cualquier orden y generar el mismo resultado.

- **Visión pesimista,** es posible que una saga lea datos sucios mientras otra saga ejecuta una transacción compensable para revertir la operación. La visión pesimista reordena la saga para que los datos subyacentes se actualicen en una **transacción reintentable**, lo que elimina la posibilidad de una lectura de datos sucios.

Recomendaciones para su implementación

Entre las contramedidas sugeridas para reducir o evitar anomalías se incluyen:

- **El valor de relectura** comprueba que los datos no se hayan modificado y, a continuación, actualiza el registro. Si el registro ha cambiado, los pasos se anulan y la saga puede reiniciarse.

- **Un archivo de versión** registra las operaciones en un registro a medida que llegan y las ejecuta en el orden correcto.

- **Por valor** usa el riesgo empresarial de cada solicitud para seleccionar dinámicamente el mecanismo de simultaneidad. Las solicitudes de bajo riesgo favorecen a las sagas, mientras que las solicitudes de alto riesgo favorecen a las transacciones distribuidas.

Cuando usar SAGA

Use el patrón de SAGA cuando necesite:

- **Garantizar la coherencia de los datos en un sistema distribuido** sin acoplamiento estricto.
- **Revertir o compensar si se produce un error** en una de las operaciones de la secuencia.

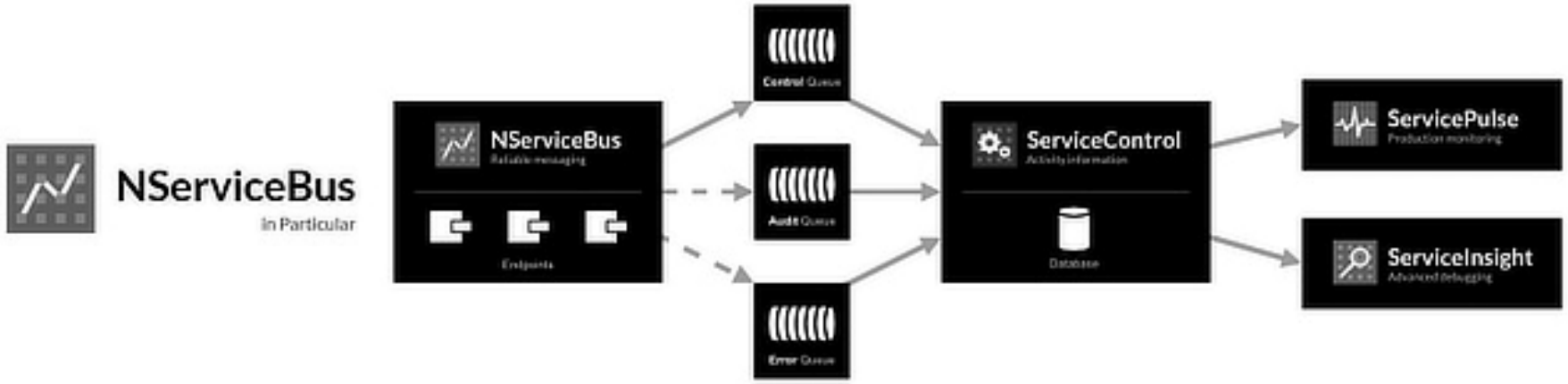
El patrón de saga es menos adecuado para:

- **Transacciones estrechamente acopladas.**
- **Transacciones de compensación que se producen en participantes anteriores.**
- **Dependencias cíclicas.**

DEMO

GALAXY
TRAINING

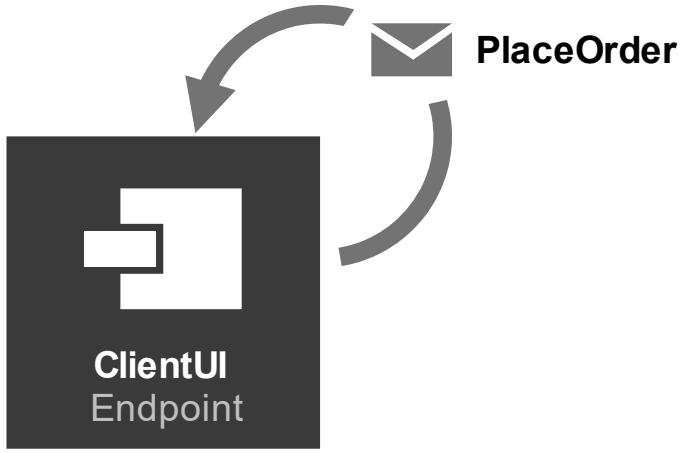
Nservices Bus



DEMO

GALAXY TRAINING

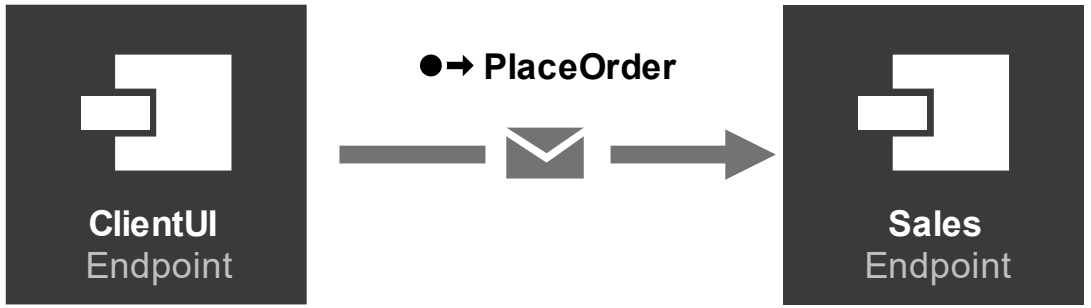
Messages



DEMO

GALAXY TRAINING

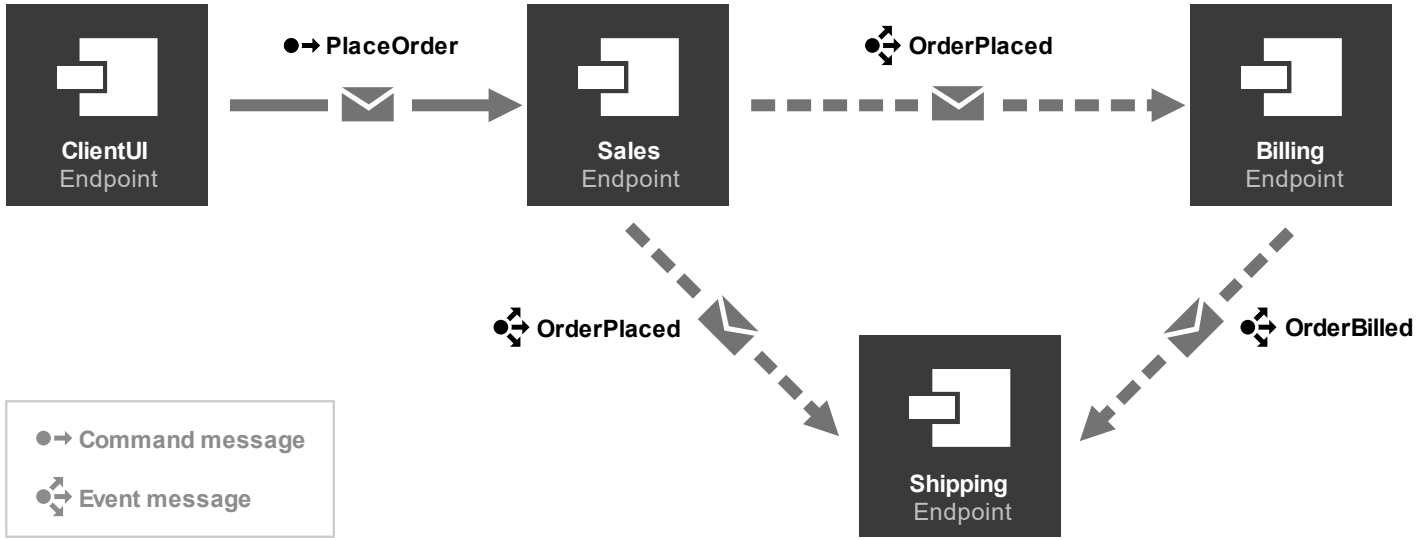
Multiple Endpoints



DEMO

GALAXY TRAINING

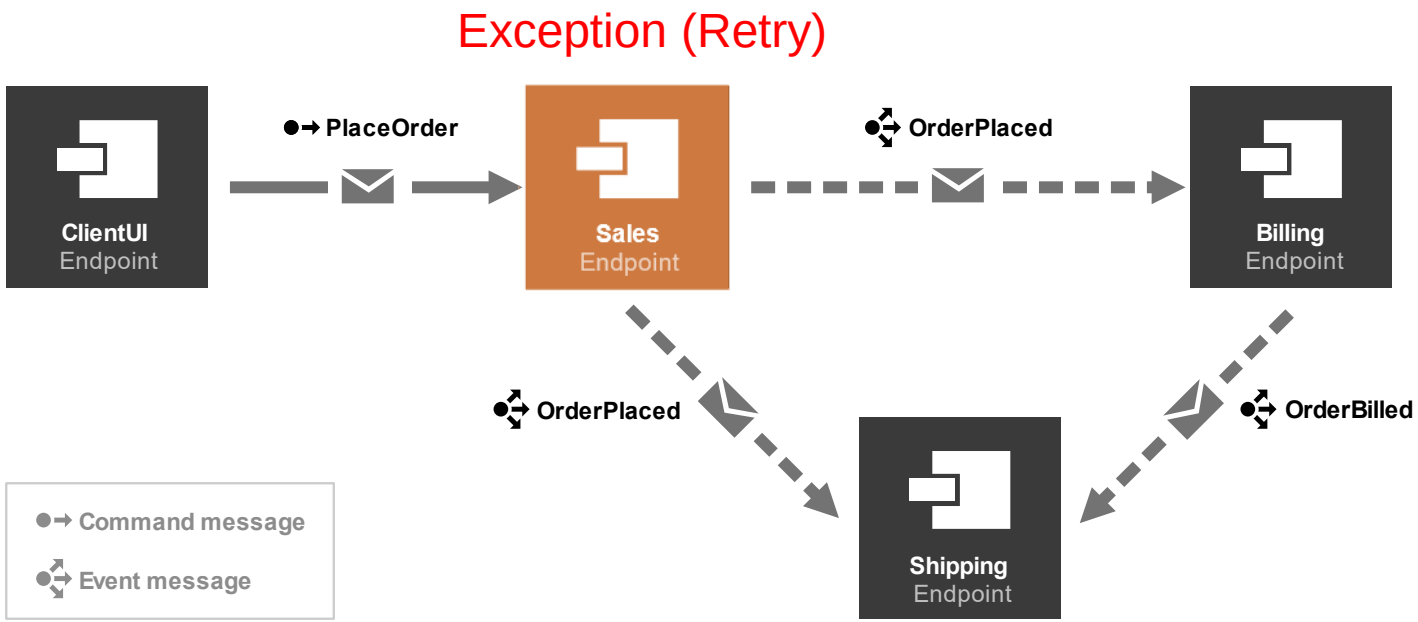
Multiple Endpoints



DEMO

GALAXY TRAINING

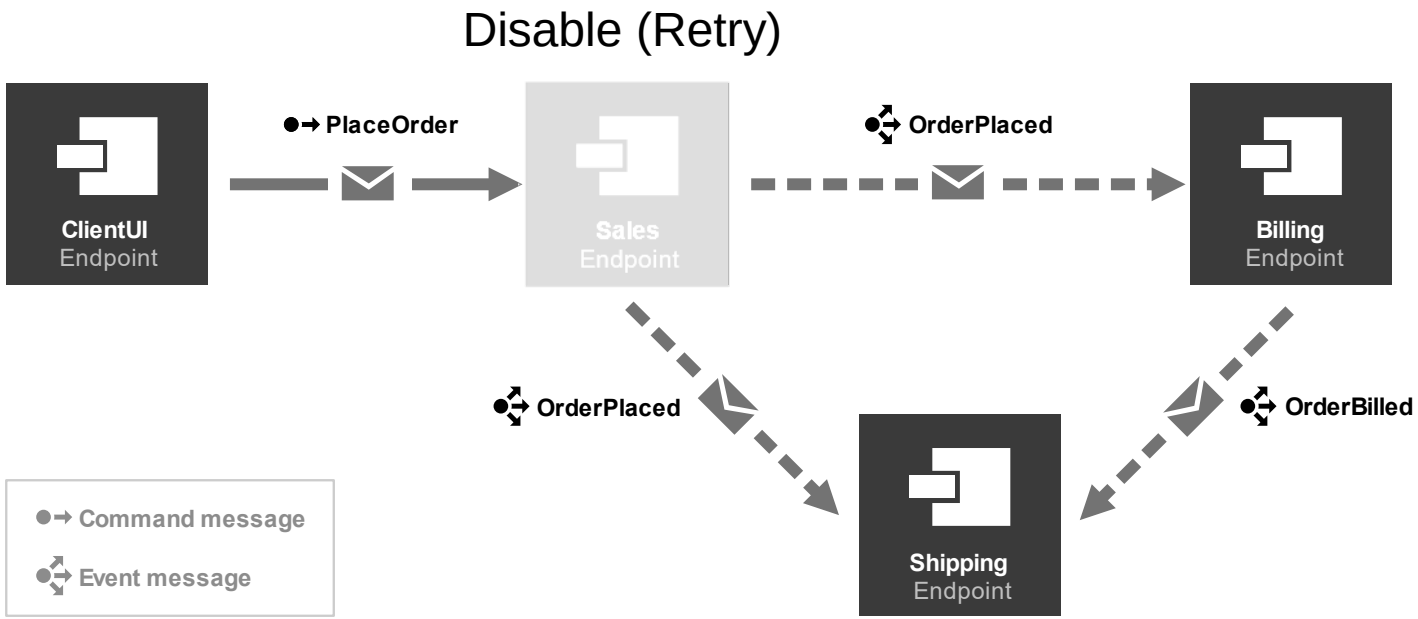
Retrying errors



DEMO

GALAXY TRAINING

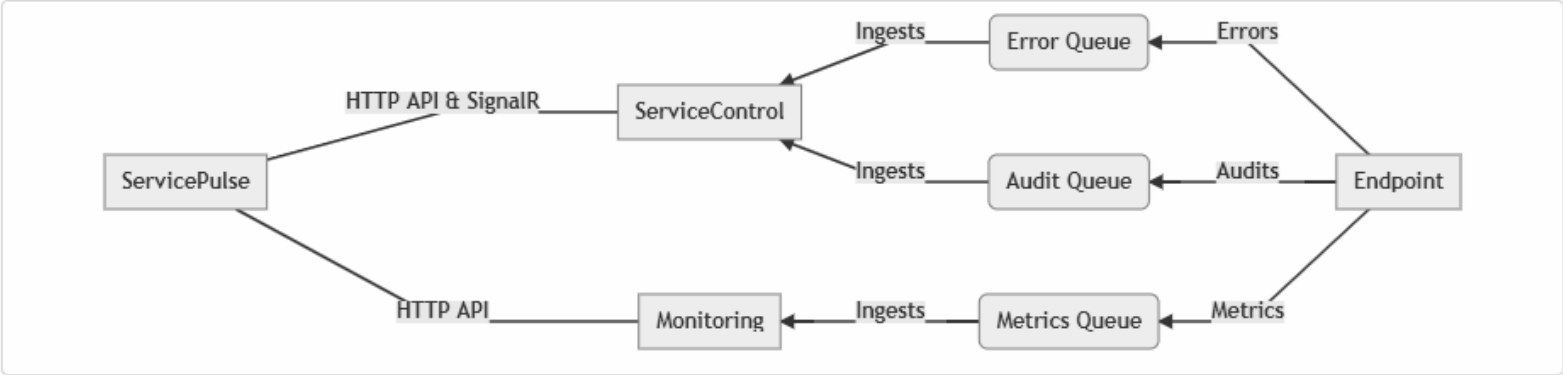
Message replay



Message replay

DEMO

ServicePulse, ServiceControl, Monitoring, and Endpoints

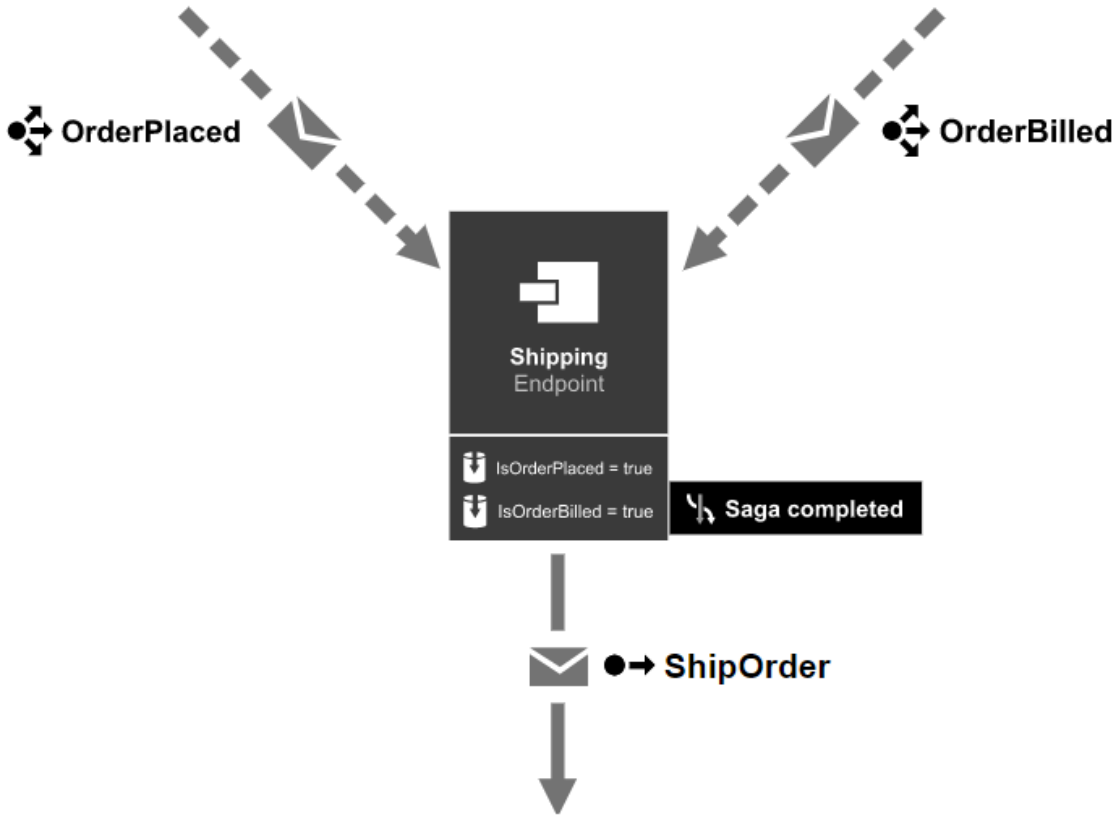


GALAXY TRAINING

DEMO

GALAXY TRAINING

Simple Saga



GRACIAS

POR SU PREFERENCIA

GALAXY
TRAINING