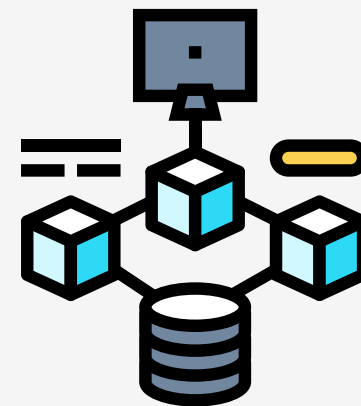


PATRÓN SAGA EN MICROSERVICIOS

PABLO DEL ÁLAMO



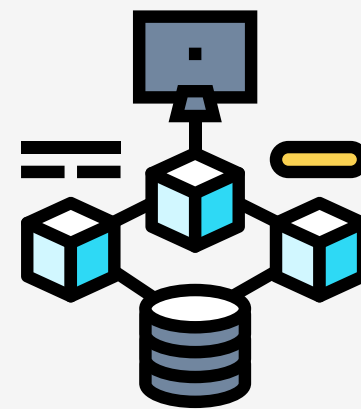
Introducción



🎯 ¿Te has preguntado cómo manejar transacciones distribuidas en microservicios, sin perder la coherencia de datos? En este documento te voy a enseñar cómo el patrón Saga soluciona este problema, ayudándote a mantener la consistencia en sistemas complejos. 🚀



¿Qué es una transacción distribuida?

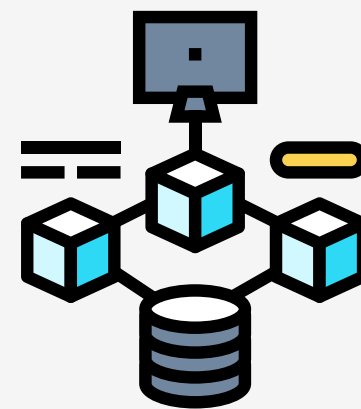


Antes de hablar de Saga, primero aclaremos qué es una transacción distribuida. 🧑🏫

Cuando trabajamos con microservicios, una "transacción distribuida" implica que varias operaciones, cada una gestionada por un servicio diferente, deben ejecutarse para completar una tarea. Esto genera un desafío: ¿Cómo garantizamos que si una parte falla, no dejemos el sistema en un estado inconsistente? 💡



Introducción al Patrón Saga

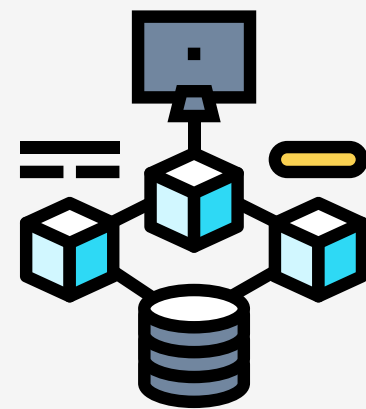


El patrón Saga es una secuencia de transacciones coordinadas de tal manera que, si alguna falla, el sistema ejecuta operaciones compensatorias para deshacer los cambios anteriores. 🎯

El objetivo: consistencia eventual. Esto significa que todos los datos llegarán a estar en un estado coherente, aunque no de inmediato.

El SEC (Service Execution Coordinator), será el servicio encargado de coordinar las transacciones. Es el encargado de decir a cada microservicio qué hacer y cuando.





Funcionamiento básico de Saga

En una arquitectura Saga, cada paso de una transacción se ejecuta como una acción local en un microservicio. Si un paso falla, una transacción compensatoria deshace lo que se hizo en los pasos anteriores. 🔄

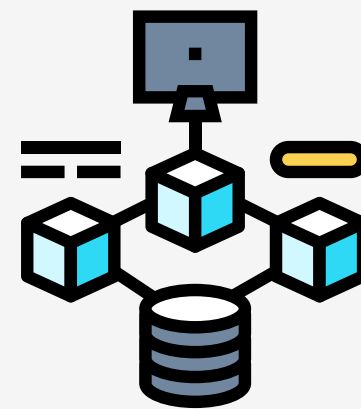
Ejemplo:

1. Creación de un pedido (Servicio A).
2. Deducción de inventario (Servicio B).
3. Procesamiento de pago (Servicio C).

Si el pago falla, entonces el inventario es restaurado y el pedido es cancelado. 💸



Tipos de implementación de Saga



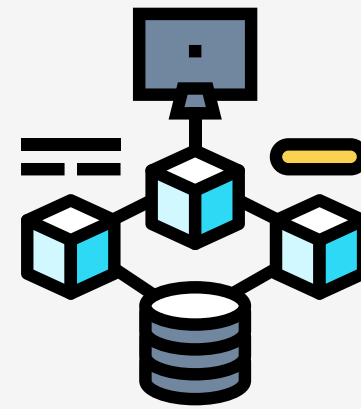
Existen dos formas principales de implementar el patrón Saga: Coreografía y Orquestación.

Ambos logran el mismo fin, pero con mecanismos muy distintos.

Vamos a profundizar en cada uno de ellos. 🔍



Coreografía



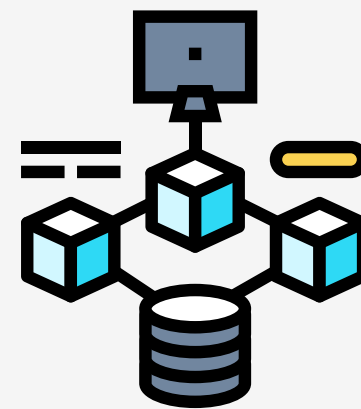
En la Coreografía, no hay un controlador central. 🌀
Cada microservicio es autónomo y conoce el evento que debe esperar para desencadenar su acción.

Cuando termina, emite un evento que otros servicios pueden escuchar para continuar la transacción.

Esta arquitectura es event-driven (dirigida por eventos). 🌐




Coreografía – Ejemplo Paso a Paso



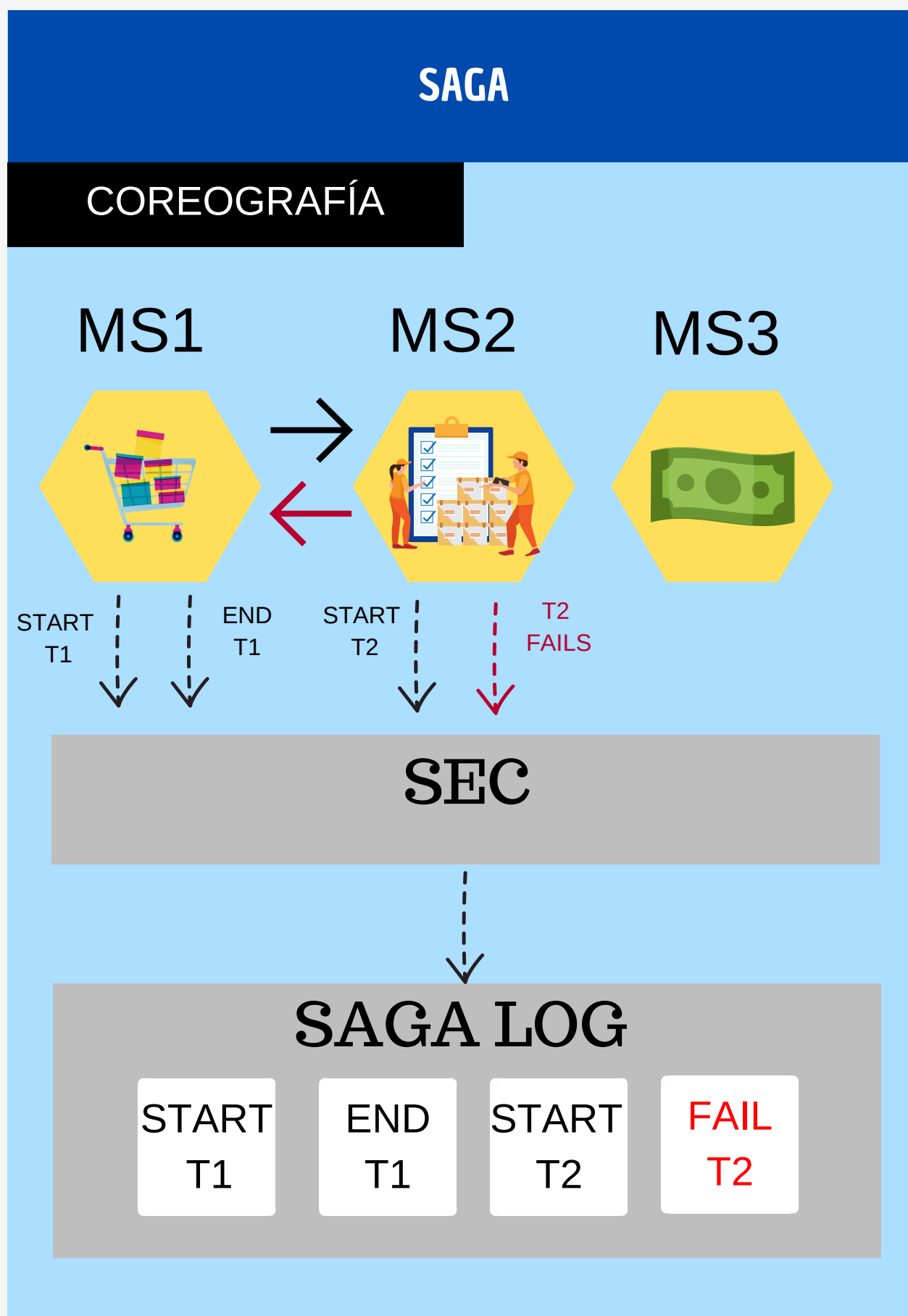
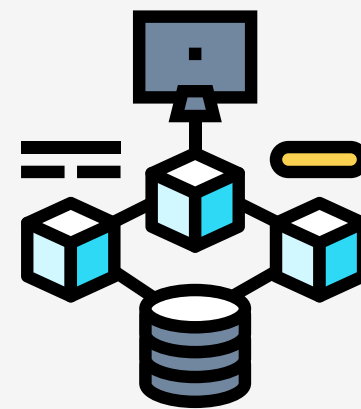
Vamos a verlo en acción:

1. Servicio A (Pedido) crea un pedido y lanza un evento: "Pedido creado".
2. Servicio B (Inventario) escucha el evento, deduce inventario, y emite un nuevo evento: "Inventario actualizado".
3. Servicio C (Pago) escucha el evento y procesa el pago.

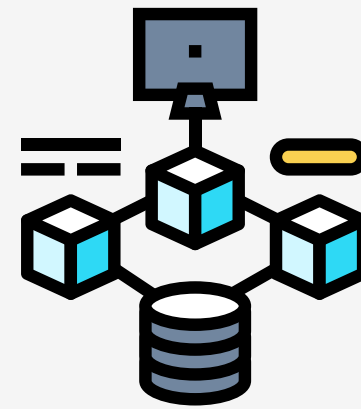
Si algo falla, cada servicio tiene su propio mecanismo de compensación para deshacer cambios. 



Coreografía – Ejemplo Paso a Paso



Ventajas de la Coreografía

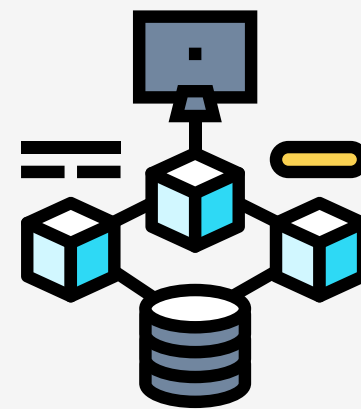


¿Por qué usar Coreografía? Aquí algunas ventajas clave:

1. Desacoplamiento: Los servicios son independientes. No dependen de un coordinador central, lo que facilita la escalabilidad. 🚀
2. Flexibilidad: Cada servicio puede evolucionar sin afectar a los demás, solo tiene que entender los eventos. 🔧



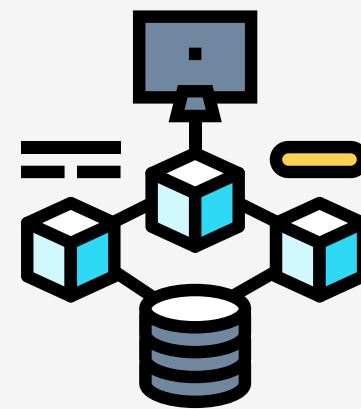
Retos de la Coreografía




- Complejidad en el manejo de eventos: A medida que crecen los microservicios, puede ser difícil rastrear qué servicio emite qué evento. 🕵️
- Baja trazabilidad: Si algo sale mal, depurar la causa del fallo puede ser complicado. 👁️👁️



Escenarios donde usar Coreografía

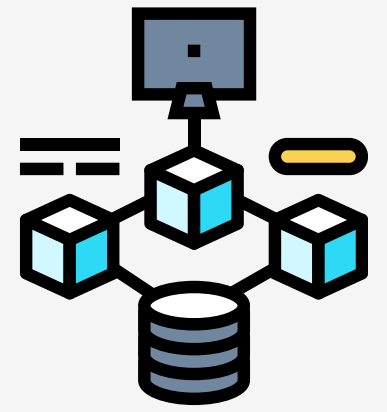


Algunos ejemplos:

- E-commerce: Cada microservicio, como inventario, pagos y envíos, puede operar de forma independiente y notificar eventos a los demás.
- Sistemas de mensajería: Donde el sistema está basado en eventos y es necesario escalar rápidamente. 



Orquestación

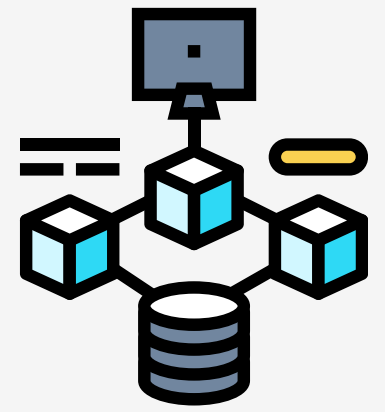


En la Orquestación, existe un servicio central llamado orquestador que controla y coordina todo el flujo de transacciones. 🎵

Este orquestador envía comandos a cada microservicio en la secuencia correcta y maneja los fallos para ejecutar las compensaciones cuando es necesario.



Orquestación – Ejemplo Paso a Paso

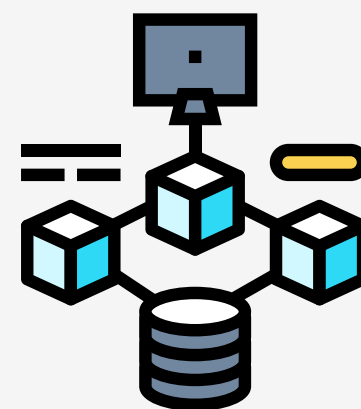


Ejemplo con orquestación:

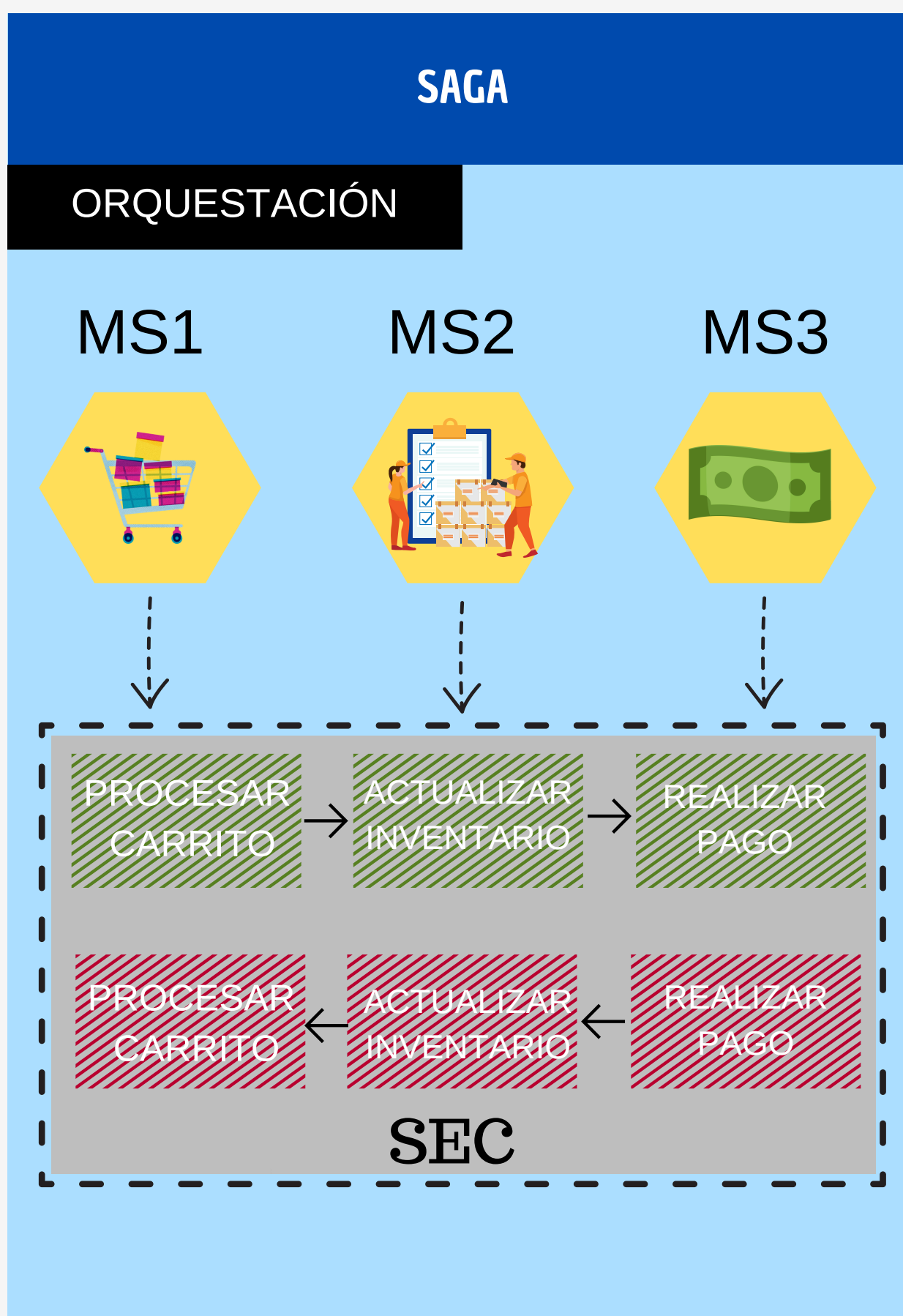
1. Orquestador inicia la Saga enviando un comando a Servicio A (Pedido).
2. Servicio A responde con el éxito/fallo. Si es éxito, el orquestador le dice a Servicio B (Inventario) que actualice su stock.
3. Cuando Servicio B confirma, el orquestador le dice a Servicio C (Pago) que cobre al cliente.

Si el pago falla, el orquestador invoca las compensaciones: revertir el inventario y cancelar el pedido. 🚦

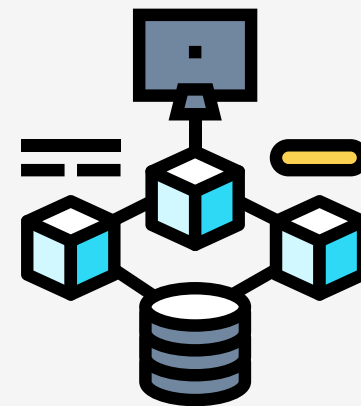






Orquestación – Ejemplo Paso a Paso



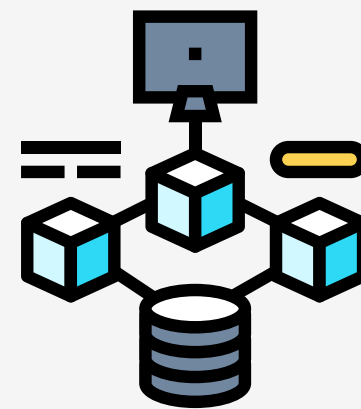
Ventajas de la Orquestación



- Control centralizado: El flujo de trabajo es más claro, porque hay un único punto de control (el orquestador). 
- Facilidad a la hora de depurar: Como el orquestador maneja todo, es más fácil rastrear dónde ocurre un fallo. 



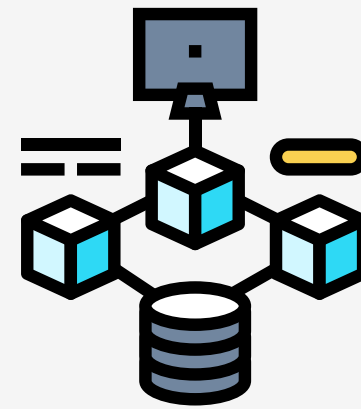
Retos de la Orquestación



- Punto único de fallo: Si el orquestador falla, toda la saga podría quedar interrumpida. ⚠️
- Mayor acoplamiento: Los microservicios dependen de este coordinador, lo que podría reducir la flexibilidad. 🤔



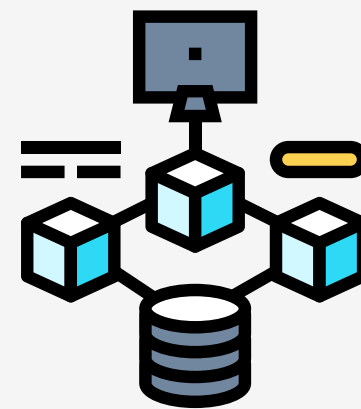
Escenarios donde usar Orquestación



- Procesos complejos: Cuando hay varias reglas de negocio que necesitan ser manejadas en secuencia, como en sistemas bancarios o de seguros. 🏛️
- Flujos bien definidos: Cuando sabes exactamente qué microservicios deben ejecutarse y en qué orden. ✓

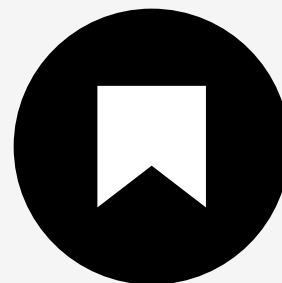
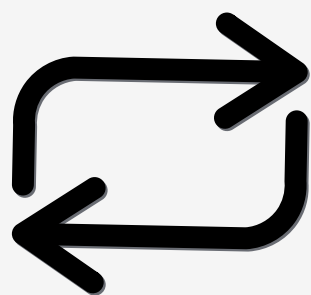


Comparación Final – Coreografía vs Orquestación



- Coreografía: Es más flexible y escalable, pero puede volverse difícil de gestionar a medida que el número de eventos crece.
- Orquestación: Es más controlada y trazable, pero introduce un punto único de fallo y puede aumentar el acoplamiento.





¿Te ha gustado el contenido? Dale like, comparte o guárdalo para tenerlo a mano más adelante, ayuda mucho y se agradece un montón 😊

