

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



How to Use Saga Pattern in Microservices

Using the Saga pattern in Microservice transactions



Chameera Dulanga · Follow

Published in Bits and Pieces · 6 min read · Oct 13, 2021



1K



2



Using the microservices architecture has many benefits. It has become the norm for many large-scale applications. However, Microservices also comes with several challenges. One such challenge is handling transactions that span across multiple services.

So, in this article, I will discuss how we can overcome this by using Saga Pattern.

Why We Need Saga Pattern?

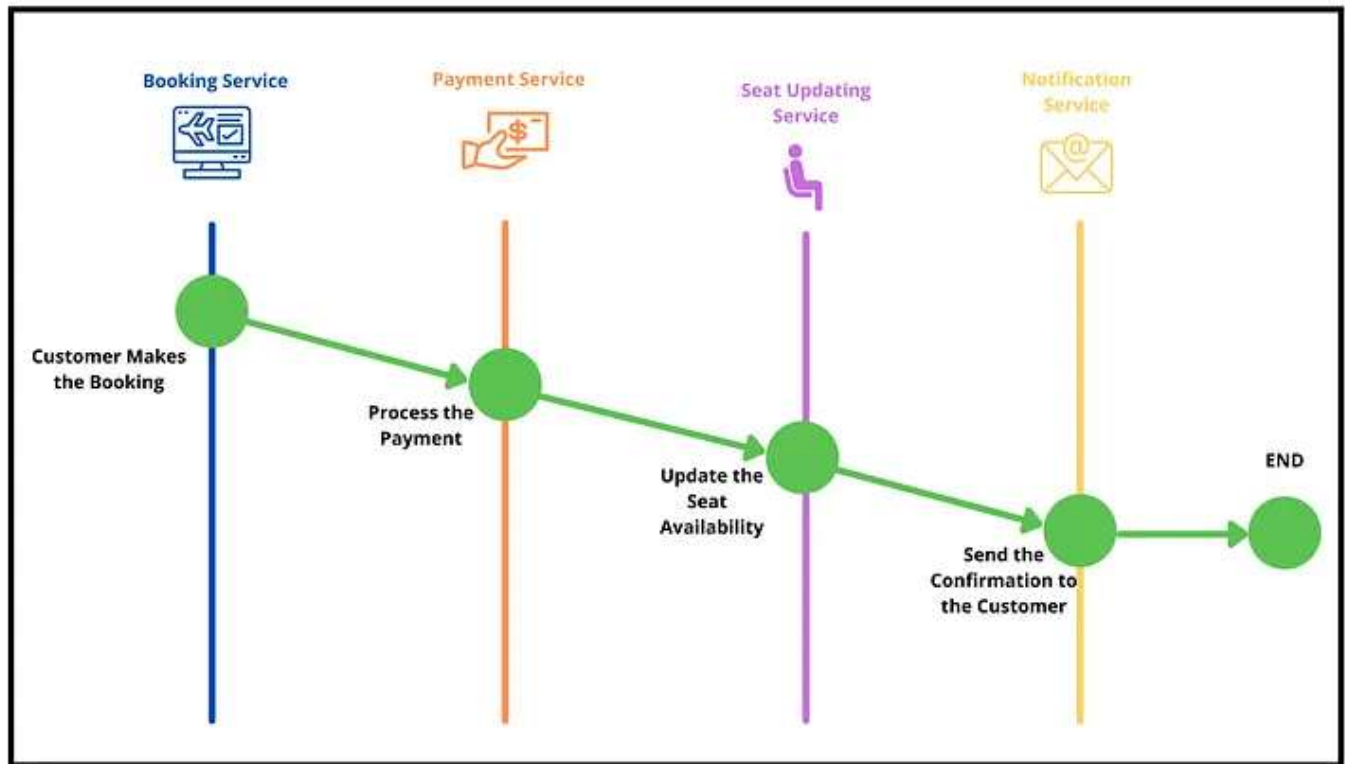
The Microservice architecture allows us to maintain applications services and their databases separately. However, with the complexity of modern-day requirements, it is common to have data changes that need to propagate across multiple services.

For example, let's consider a simple online booking ticket system built with Microservices. There are separate Microservices for;

- Ticket booking.
- Payment processing.
- Updating available seats.
- Sending confirmations to customers.

Suppose a customer makes a booking. It requires invoking several Microservices in sequence to complete the flow starting from booking, payment, reserve seats, and send confirmation.

But, what happens if any of these steps fail? We somehow need to roll back any previous steps to maintain data integrity.

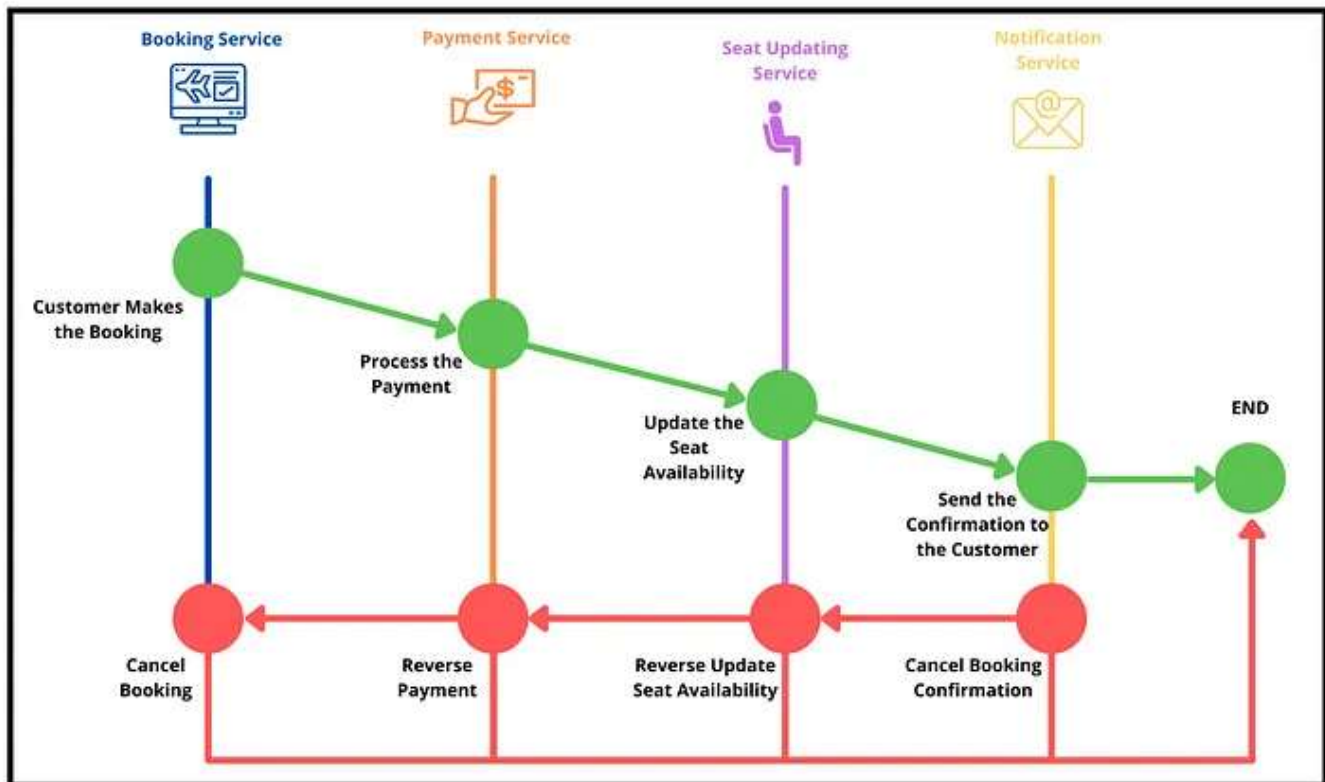


However, since each Microservice performs local transactions, it is hard to maintain ACID (Atomicity, Consistency, Integrity, Durability) properties across Microservices.

Therefore, we need a centralized communication and coordination mechanism to ensure all the transactions are completed successfully, and that's where the Saga pattern comes in.

Introduction to Saga Pattern

The Saga pattern manages transactions that span across multiple Microservices using a sequence of local transactions. The following diagram contains the rollback events in red color, which is a part of the SAGA workflow.



These events are triggered by the Saga Execution Controller in case of a failure happens to ensure the system's stability.

What is Saga Execution Controller?

Saga execution controller (SEC) is the centralized component that controls the local transactions and rollback events.

It tracks all the events of distributed transactions as a sequence and decides the rollback events in case of a failure.

Also, the SEC makes sure that rollback events do not have any additional effect other than reversing the local transactions.

Note: SEC internally uses a log named Saga log to keep track of all transactions

I think now you have a high-level understanding of what the Saga pattern is and how it works. To get a better understanding, let's see how we can implement it.

Implementing Saga Pattern

There are 2 approaches to implement the Saga pattern, Choreography-Based Saga and Orchestration-Based Saga. So, let's see how these 2 approaches are different from each other and how they work.

1. Orchestration-Based Saga

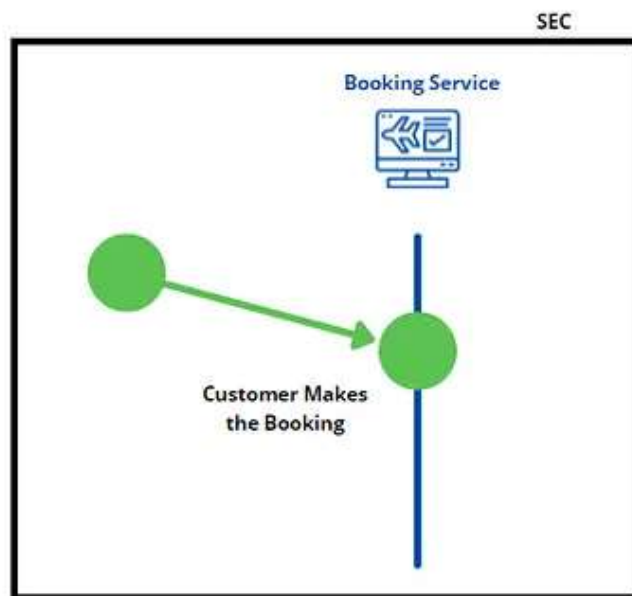
In Orchestration-Based Saga, a single orchestrator (arranger) manages all the transactions and directs services to execute local transactions.

The orchestrator acts as a centralized controller of all these local transactions and maintains the status of the complete transaction.

Let's consider the same example I explained earlier and break down Orchestration-Based Saga into steps.

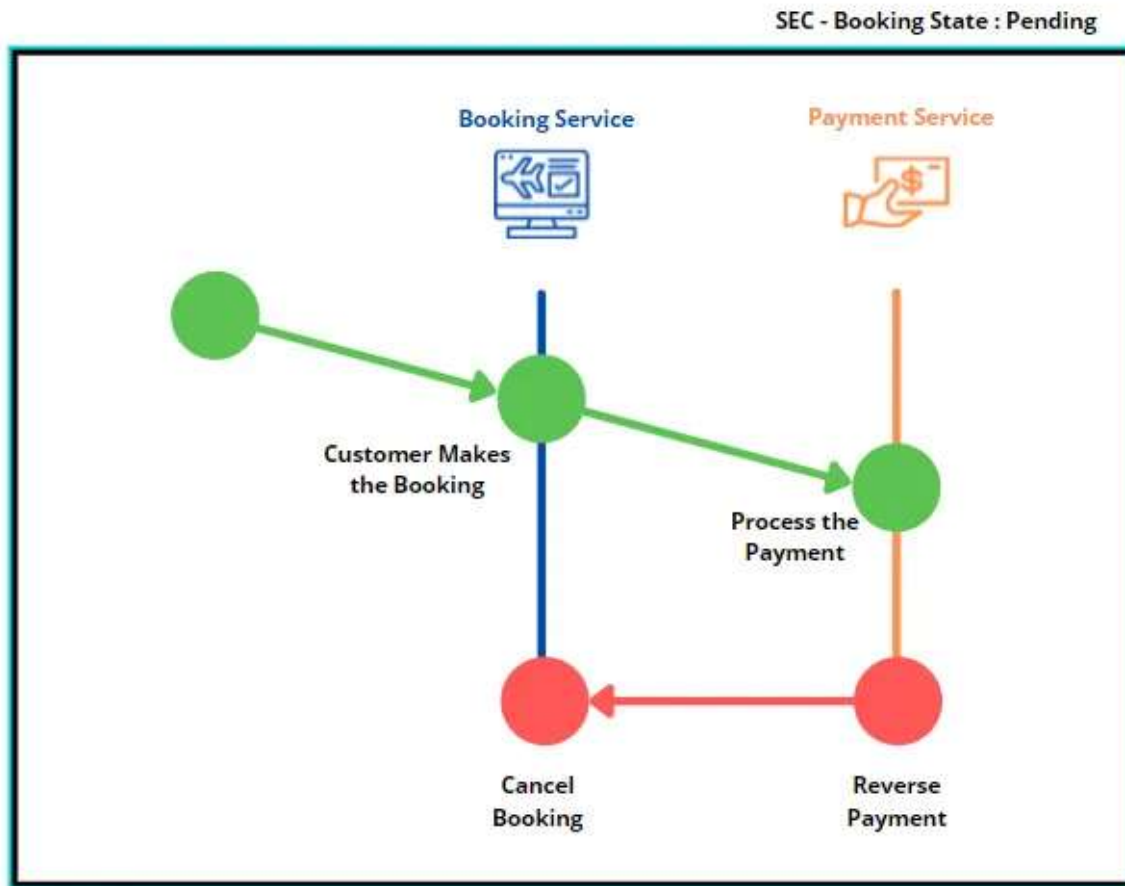
Step 1 — User makes the booking request

When a user makes a new booking request, the booking service receives the POST request and creates a new Saga orchestrator for the booking process.



Step 2 — Orchestrator creates a new booking

Then the orchestrator creates a new booking in the pending state and sends the payment process command to the payment service.

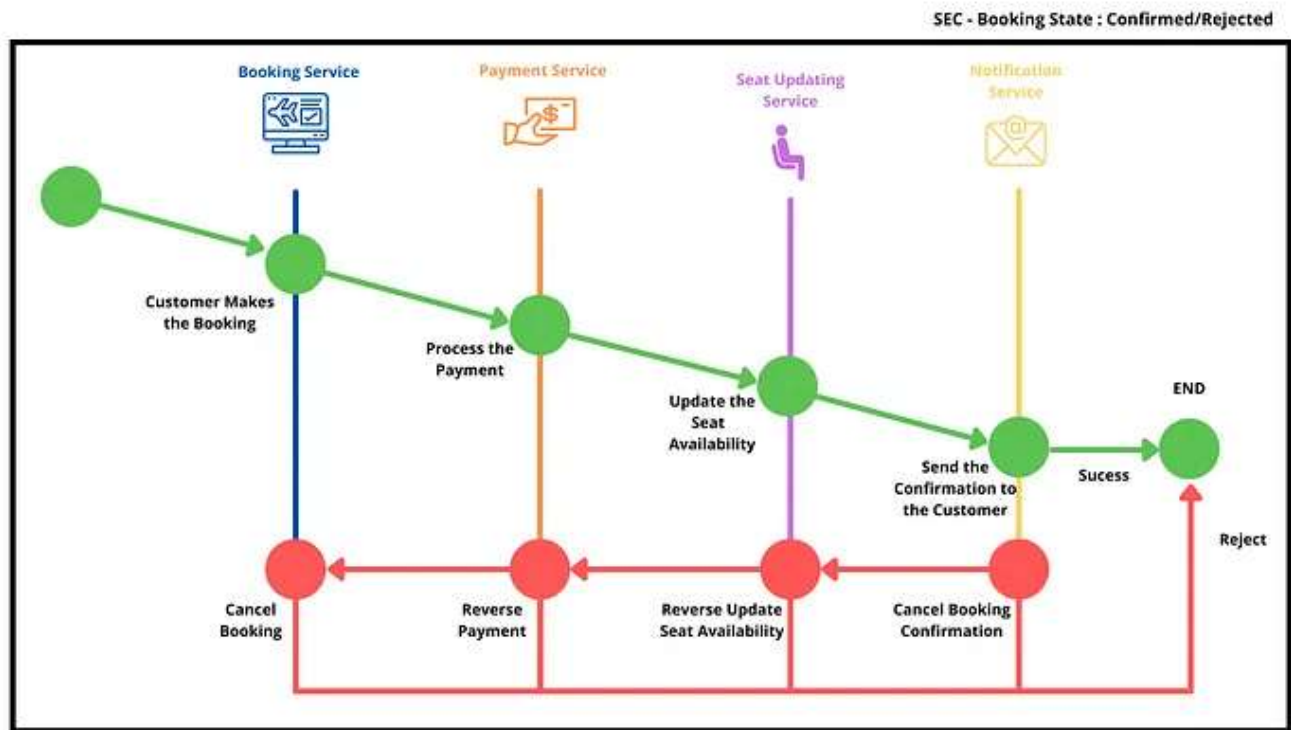


Step 3 — Executes all the services accordingly

After the payment is processed successfully, the orchestrator will call the seat updating service. Likewise, all the services will be called one by one, and the orchestrator will be notified whether the transactions are succeeded or failed.

Step 4 — Approve or reject the booking

After each service completes its local transaction, they inform the transaction status to the orchestrator. Based on that feedback, the orchestrator can approve or reject the booking by updating its state.



Orchestration-Based Saga is more simple compared to Choreography-Based Saga, and it is most suitable for situations like,

- When there are already implemented Microservices.
- When a large number of Microservices participate in a single transaction.

2. Choreography-Based Saga

In Choreography-Based Saga, all the services that are part of the distributed transaction publish a new event after completing their local transaction.

The Choreography-Based Saga approach does not have an orchestrator to direct and execute local transactions. Instead, each Microservice is

[Open in app](#)



Search

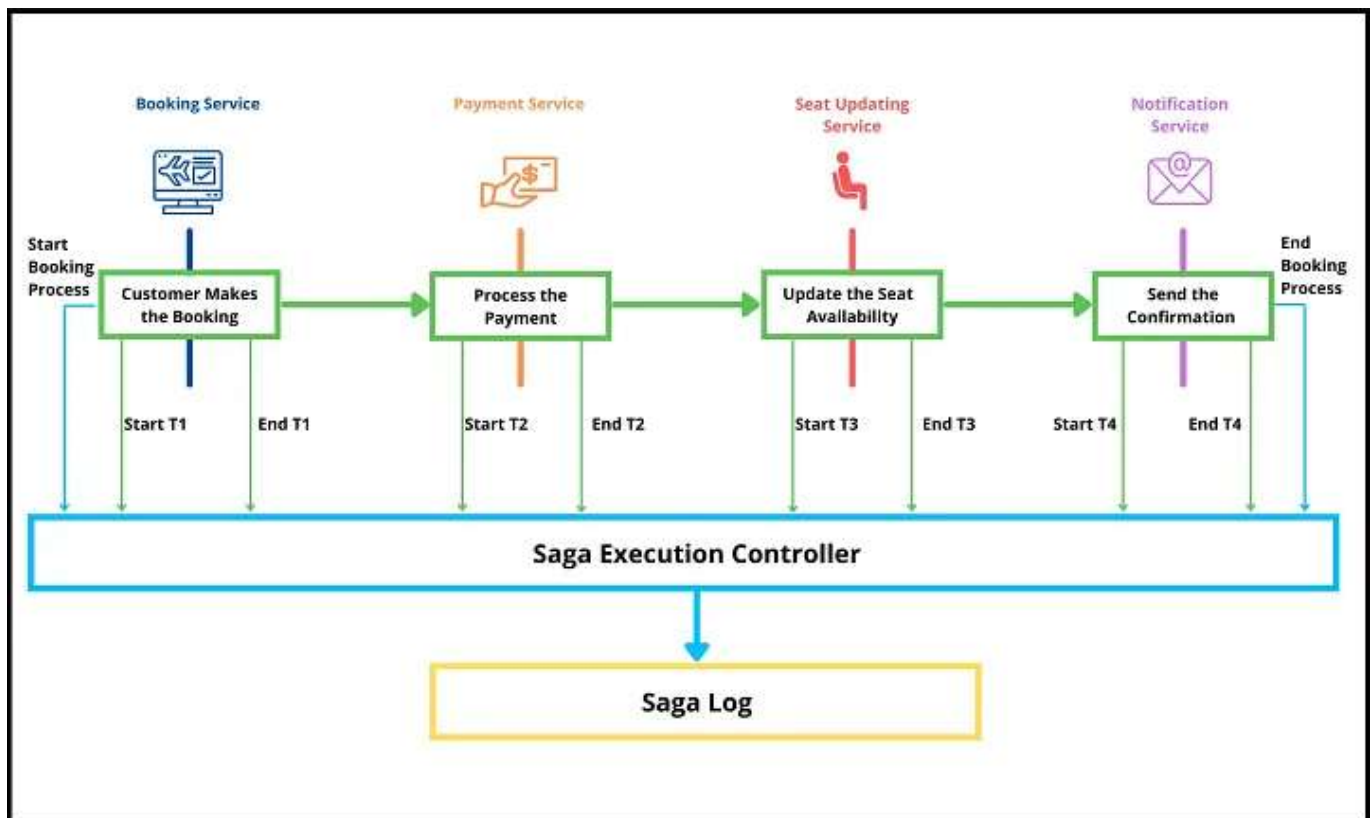
Write



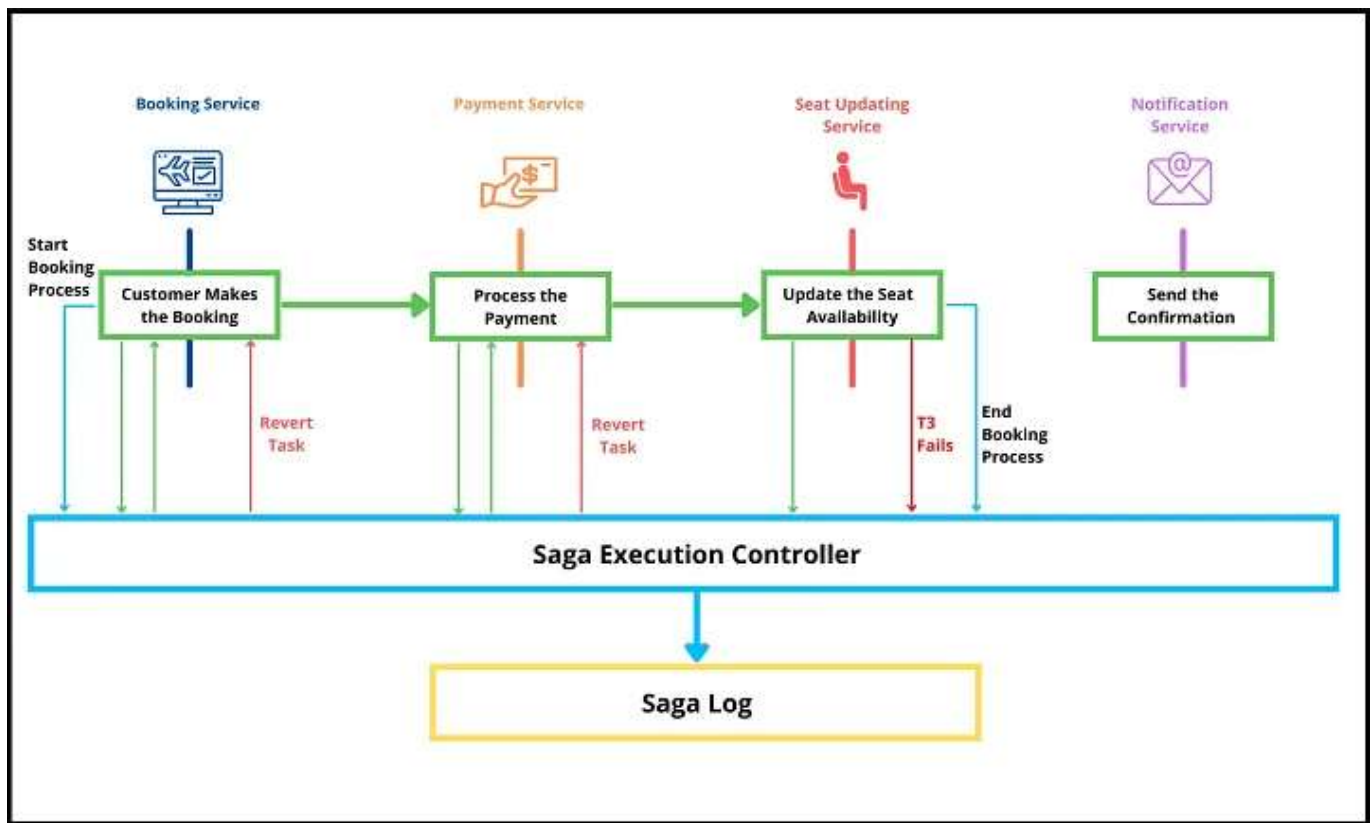
J

Saga execution controller keeps track of all these events using the SEC log and executes rollback events in case of a failure.

For example, the booking process we discussed earlier would look like this with the Choreography-Based Saga approach.



But, what would happen if the seat updating transaction fails? Then, the seat updating service will inform the SEC about the failure, and SEC will start the corresponding rollback events and ends the booking process by setting the state to fail.



As you can see, this approach is more complex than the Orchestration-Based Saga approach. So that the Choreography-Based Saga approach is more suitable for situations like,

- When you implement new Microservices from scratch.
- When a small number of Microservices participate in a single transaction.

Final Thoughts

In this article, I discussed what is Saga pattern is and different approaches for implement it.

The Saga pattern's main advantage is to maintain data consistency when transactions span across Microservices.

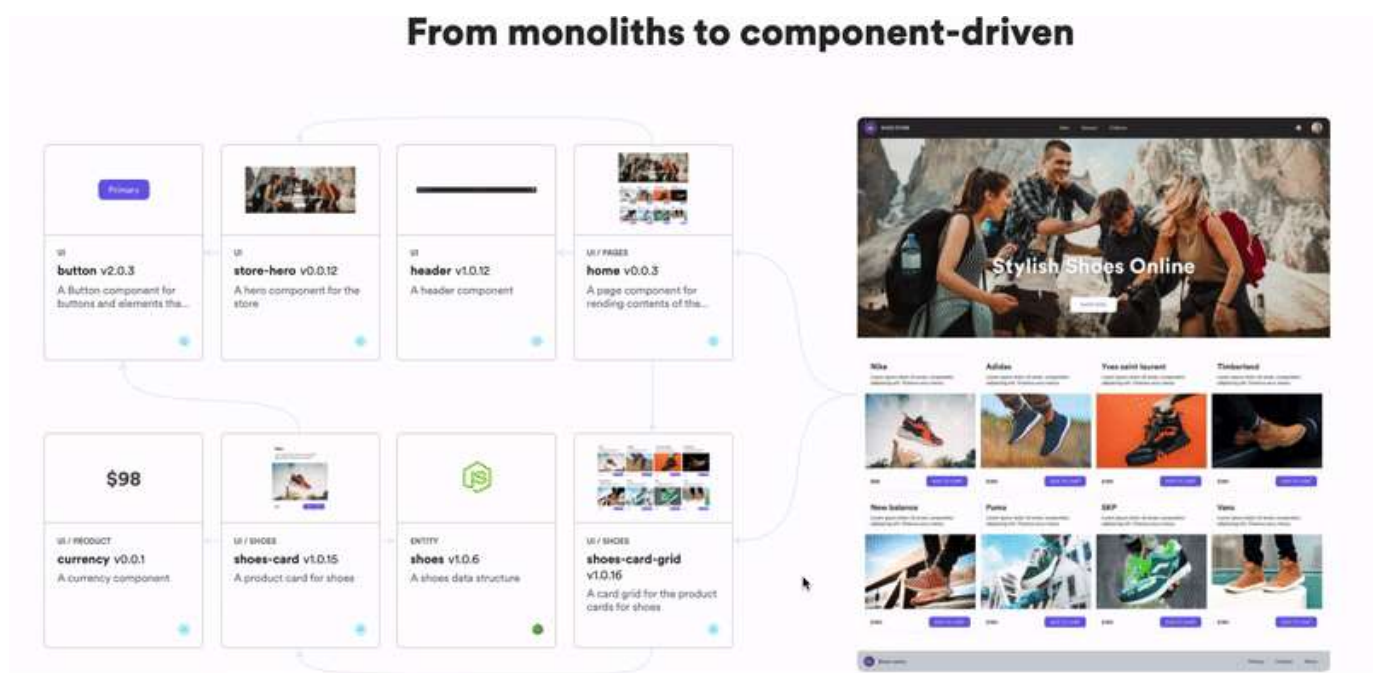
Also, frameworks like Camunda, Apache Camel, Axon Saga, and Eventuate Tram Saga help you implement the Saga pattern in practice.

Thank you for Reading !!!

Build composable web applications

Don't build web monoliths. Use Bit to create and compose decoupled software components — in your favorite frameworks like React or Node. Build scalable and modular applications with a powerful and enjoyable dev experience.

Bring your team to Bit Cloud to host and collaborate on components together, and greatly speed up, scale, and standardize development as a team. Start with composable frontends like a Design System or Micro Frontends, or explore the composable backend. Give it a try →



Learn More

How We Build Micro Frontends

Building micro-frontends to speed up and scale our web development process.

blog.bitsrc.io

How we Build a Component Design System

Building a design system with components to standardize and scale our UI development process.

blog.bitsrc.io

The Composable Enterprise: A Guide

To deliver in 2022, the modern enterprise must become composable.

blog.bitsrc.io

7 Tools for Faster Frontend Development in 2022

Tools you should know to build modern Frontend applications faster, and have more fun.

blog.bitsrc.io

Microservices

Saga

Saga Pattern

Web Development

Nodejs



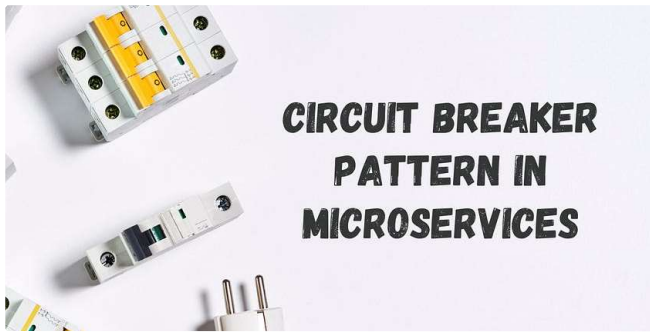
Written by Chameera Dulanga


4K Followers · Writer for Bits and Pieces

Software Engineer | AWS Community Builder (x2) | Content Manager



More from Chameera Dulanga and Bits and Pieces



 Chameera Dulanga in Bits and Pieces

Circuit Breaker Pattern in Microservices

How to Use the Circuit Breaker Software Design Pattern to Build Microservices

6 min read · Jan 11

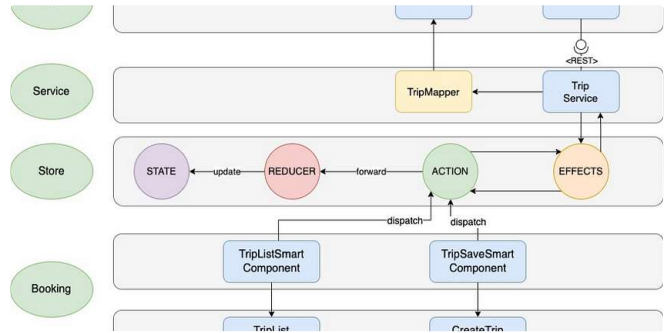


575

3



...



 Robert Maier-Silldorff in Bits and Pieces

Clean Frontend Architecture

An overview of some of the principles associated with a clean frontend architecture...

6 min read · Jul 14

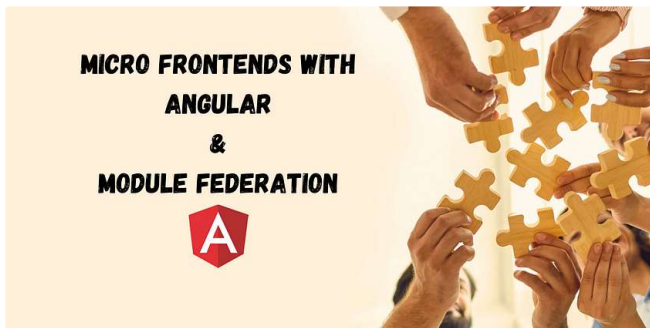


2K

17



...



 Thamodi Wickramasinghe in Bits and Pieces

You've Been Building Angular Apps Wrong!

Break Free from Monolithic Limits: Build Better with Micro Frontends in Angular

10 min read · Nov 17

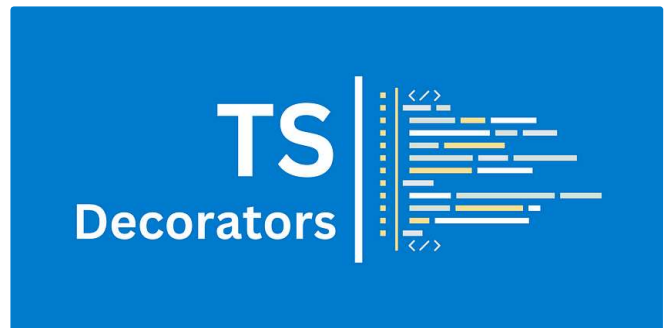


746

7



...



 Chameera Dulanga in Bits and Pieces

Using TypeScript Decorators in Practise

TypeScript 5.0 Decorators

6 min read · Nov 24



375

2

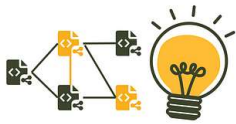


...

[See all from Chameera Dulanga](#)[See all from Bits and Pieces](#)

Recommended from Medium

REUSING CODE. Microservices



Ashan Fernando in Bits and Pieces

Is it safe to share code between Microservices?

Understanding the Fundamentals of Code Sharing in Microservices

6 min read · Nov 20



356



2



Ian Kiprono in Stackademic

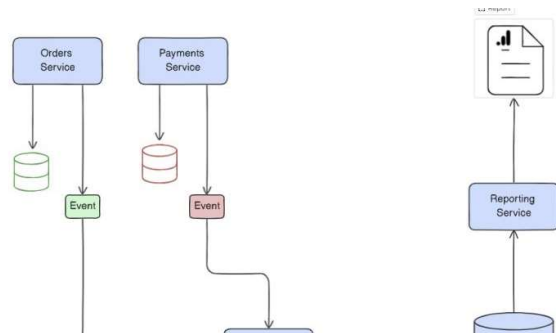
Handling Reporting on Microservices Architecture

Ask around and most developers will tell you how frightening Microservices are. It should...

2 min read · Oct 26



52



Lists



Coding & Development

11 stories · 319 saves



General Coding Knowledge

20 stories · 677 saves



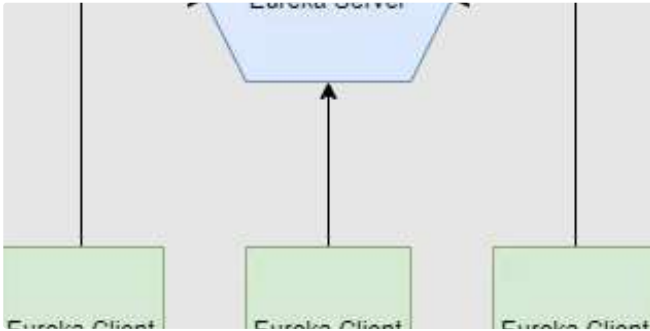
Stories to Help You Grow as a Software Developer

19 stories · 630 saves



Tech & Tools

15 stories · 109 saves



seyhmusaydogdu

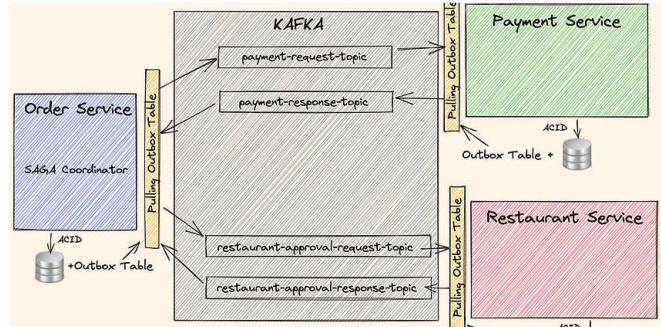
Eureka Server and Discovery Client with Spring Boot 3.1.1

In this section, With the latest version of Spring Boot, 3.1.1, we will create the most use...

3 min read · Jun 29



53



The Java Trail

Consistency in Microservices: Transactional Outbox Pattern

In a microservices architecture, different services often handle different aspects of...

6 min read · Dec 2



150



1

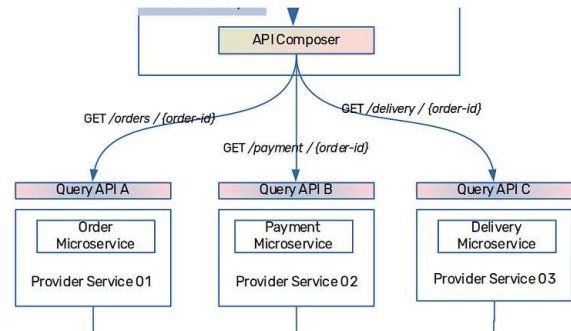


Dr. Ernesto Lee

Building a .NET 7 Microservice with Monitoring and Centralized...

Introduction

4 min read · Sep 13



Crishantha Nanayakkara

Microservices Patterns: API Composition Pattern

Microservices Patterns Series—Part 07

4 min read · Jul 8



65



See more recommendations