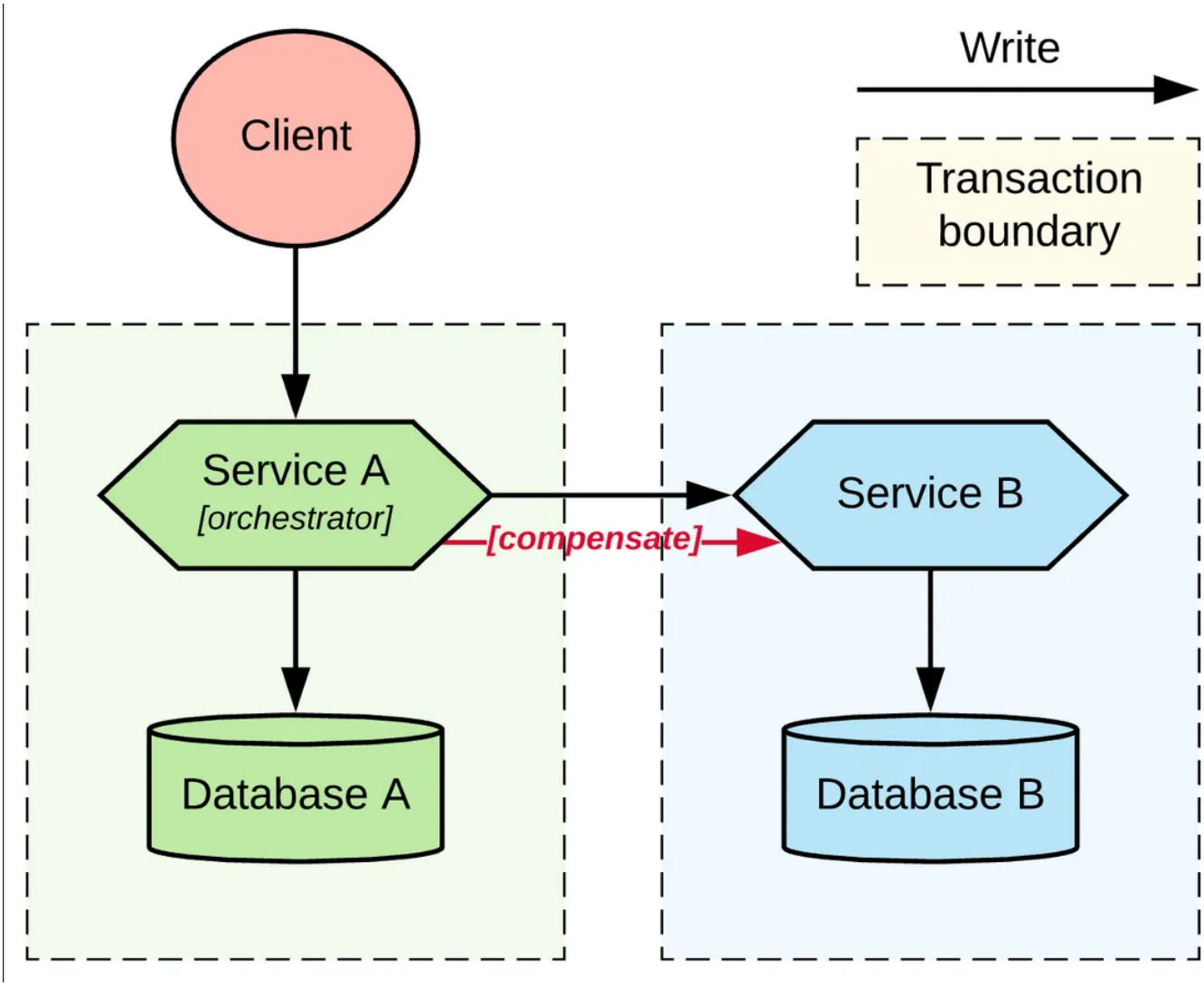
Member-only story

How to manage transactions in Distributed Systems and Microservices?

In this article you will learn about 3 ways to manage transactions in distributed systems and microservices such as two-phase commit, SAGA, and event sourcing.

Soma · Following
Published in Javarevisited
9 min read · Mar 21



Hello guys, if you are preparing for Microservices interview question then you may know that *how do you manage transactions in a distributed system or Microservices?* is one of the **popular Microservice interview question** and why not, its one of the important concept.

In the past, I have shared *essential Microservices design principles and best practices* as well popular Microservices design patterns and in this article, I will answer this question and also tell you different ways to handle transactions in distributed system.

If you have worked in Microservices architecture, then you know that a single business transaction often spans to multiple microservices and it that's why its challenging to ensure data consistency and manage transactions.

> A poorly managed transaction can lead to data inconsistencies, lost updates, or duplicate records, which can have serious consequences for the business.

Therefore, it is crucial to have a robust transaction management system in place to ensure the reliability of the system.

In this article, we will explore v**arious strategies and best practices for managing transactions in distributed systems and microservices.** We will discuss the challenges associated with distributed transactions and how to overcome them, including two-phase commit and saga patterns.

Additionally, we will cover the different transaction management mechanisms provided by popular microservices frameworks, such as Spring Cloud and Apache Kafka.

By the end of this article, you will have a clear understanding of how to manage transactions in your distributed system and ensure the reliability and consistency of your data.

By the way, if you haven't joined Medium yet then I also suggest you to **join Medium** to read your favorite stories without interruption and learn from great authors and developers. You can **join Medium here,** its also not costly, just $5 per month.

Join Medium with my referral link - Soma

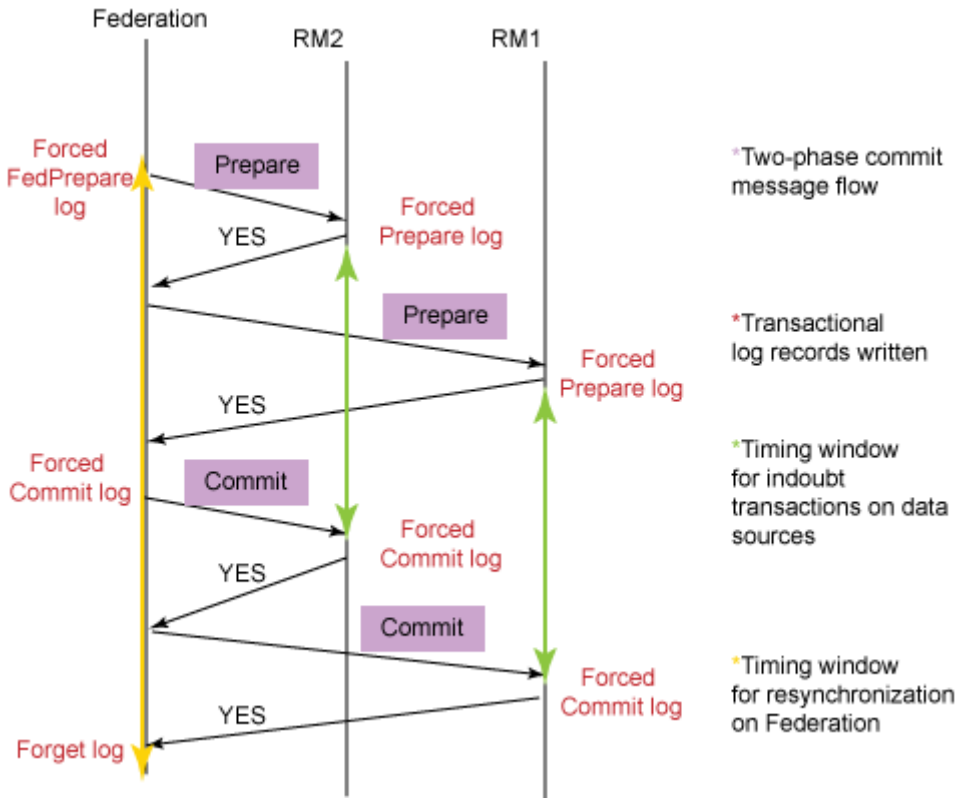
3 Ways to Manage Transactions in Distributed Systems and Microservices?

As I said, managing transactions in distributed systems and microservices can be a complex task due to the distributed nature of the system. In real world, Transactions are critical in ensuring data consistency and integrity in distributed systems where multiple services need to collaborate and share data.

One of the most popular approach to managing transactions in distributed systems is to use a **two-phase commit protocol (2PC)**. In this protocol, a **transaction coordinator** is responsible for ensuring that all participants in the transaction agree to commit the transaction.

The coordinator first sends a “prepare to commit” message to all participants, and if all participants respond positively, it sends a “commit” message to all participants.

If any participant responds negatively, the coordinator sends an “abort” message to all participants, and the transaction is rolled back.



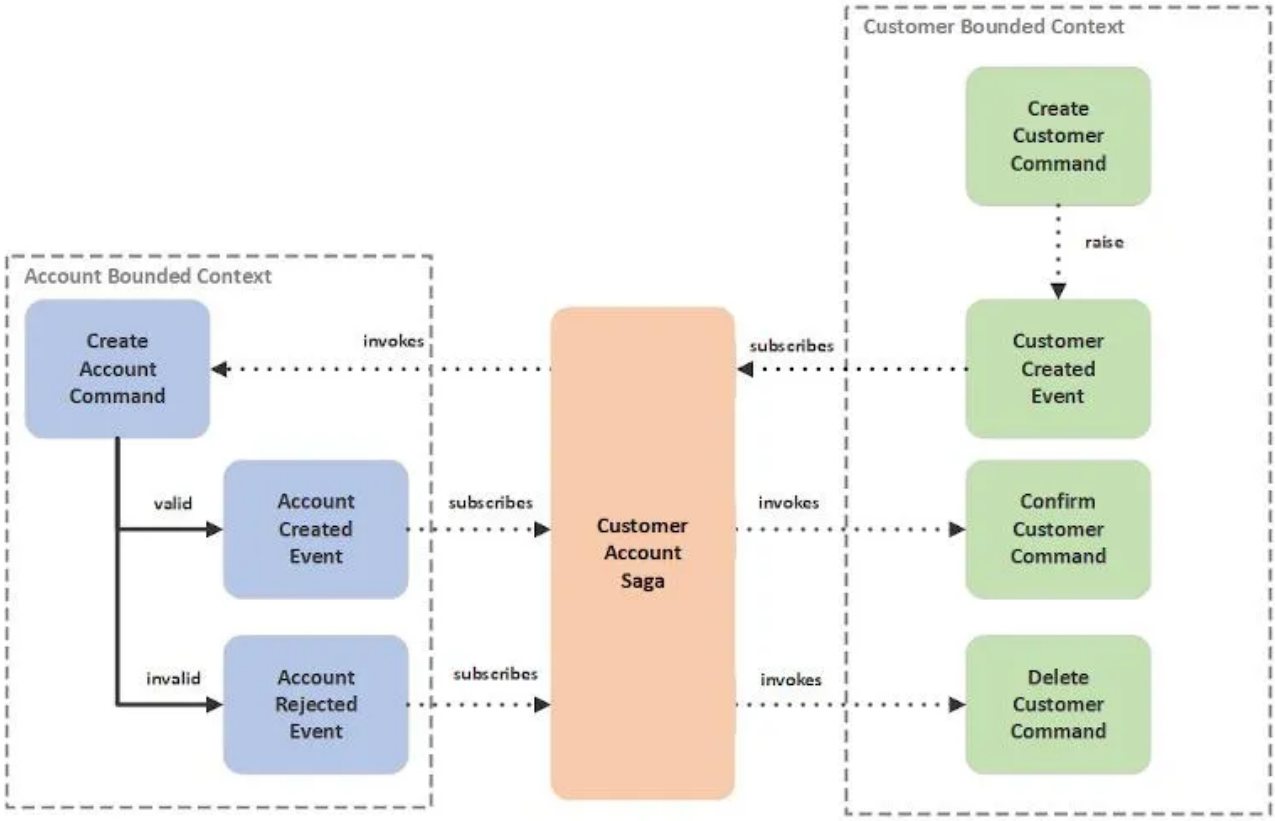
However, **2 Phase Commit can be a performance bottleneck** and can lead to reduced availability in the system. An alternative approach is to use a **compensation-based transaction protocol**. In this protocol, each participant in the transaction is responsible for compensating for any effects of the transaction if it fails.

This approach is more flexible and can be faster and more scalable than 2PC, but it requires careful design of the compensating actions.

Other techniques for managing transactions in distributed systems include Saga Pattern and Event sourcing. **Saga** is a transactional pattern that uses a sequence of local transactions to achieve a global transaction.

On the other hand, **Event sourcing** involves storing all changes to the system's state as a sequence of events, rather than the current state, which can simplify transaction management.

In Microservices architecture, it is essential to ensure that each Microservice's transactional boundaries are correctly defined to avoid inconsistent data updates across multiple services.



It is also essential to use a transaction manager that can coordinate distributed transactions across multiple microservices.

In short, managing transactions in distributed systems and microservices can be challenging, but using appropriate transaction management techniques, such as 2PC, compensation-based protocols, Saga, and Event sourcing, can help ensure data consistency and integrity in distributed systems.

Now that you know the basics, let's look at these techniques in little bit more detail.

What is 2 Phase Commit? How does it help with distributed transaction management?

The Two-Phase Commit (2PC) is a distributed transaction protocol used to manage transactions across multiple distributed systems. It ensures that all the nodes participating in a distributed transaction agree on committing or rolling back the transaction.

In a distributed transaction, a coordinator node is responsible for managing the transaction and coordinating with all the participating nodes.

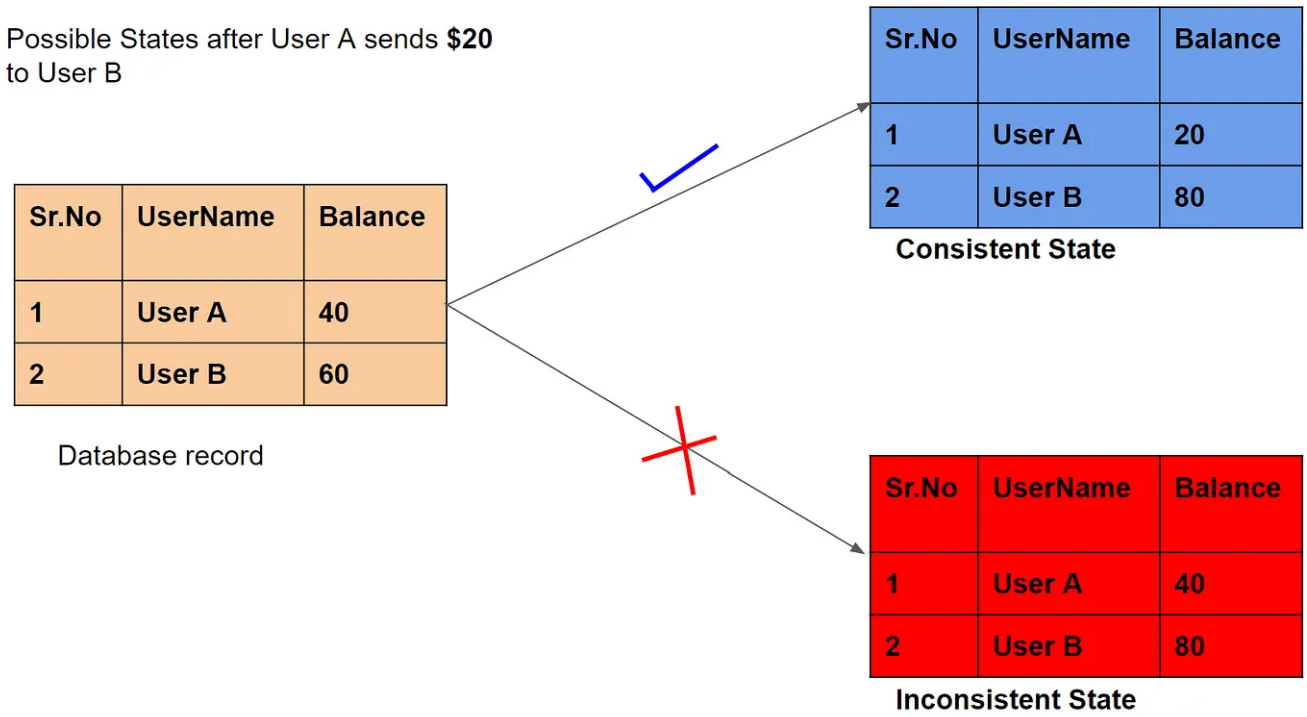
The **coordinator sends a prepare message to all the participants to confirm their readiness to commit the transaction.** If all participants are ready, the coordinator sends a commit message, and all the participants commit the transaction.

On the other hand, **if any participant is not ready or responds negatively, the coordinator sends a rollback message to all the participants**, and they all roll back the transaction.

The Two-Phase Commit protocol ensures that a transaction is either committed or rolled back in a distributed environment, even if there are communication failures or crashes. This protocol is widely used in distributed databases, messaging systems, and other distributed applications.

For example, consider a banking system that has multiple branches, and each branch has its own database. When a customer performs a transaction at one branch, it needs to be synchronized with the other branches' databases to maintain consistency.

In such cases, the Two-Phase Commit protocol can be used to ensure that the transaction is committed or rolled back consistently across all the branches.



What is SAGA Design Pattern in Microservices? How does it help with managing distributed transaction?

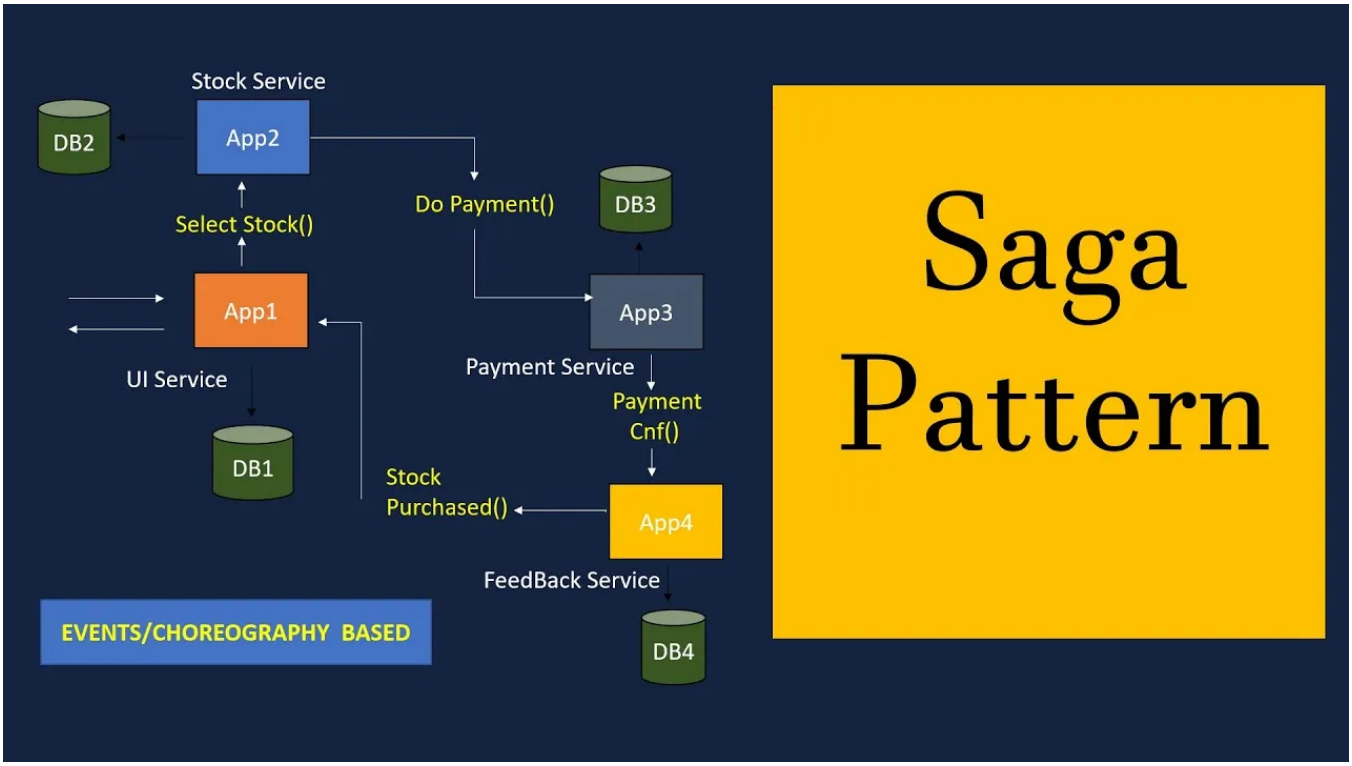
SAGA, short for "Saga Pattern," is a design pattern that helps manage distributed transactions in microservices architecture. In a microservices environment, where different services can have their own databases and transactions, the SAGA pattern provides a way to ensure transactional consistency across services.

SAGA breaks down a long-running transaction into multiple smaller transactions, each of which can be committed or rolled back independently. These smaller transactions are orchestrated by a saga coordinator, which is responsible for executing the saga in the correct order and managing compensating actions in case of failures.

The SAGA pattern can be implemented in two ways: **choreography-based** and **orchestration-based.** In choreography-based SAGA, each service is responsible for its own transactions and coordination is done through event-driven communication between services.

While, in orchestration-based SAGA, the saga coordinator is responsible for coordinating the transactions and controlling the flow of the saga.

Here is a nice diagram which explains Saga Pattern in distributed Microservices:



What is Event Sourcing pattern in Microservices? How does it help with managing distributed transaction?

Event Sourcing is a design pattern used in Microservices architecture that involves persisting the state of a system as a sequence of events rather than the current state of the system.

It helps with managing distributed transactions by providing a reliable and scalable way to handle complex business processes.

In Event Sourcing, all changes made to the state of a system are captured as events that are appended to an event log. The event log serves as the source of truth for the state of the system, allowing for easy rollback or replay of events if needed. **This approach ensures that all changes to the system are traceable, auditable, and can be easily rolled back if necessary.**

One of the key benefits of Event Sourcing is that it enables the creation of loosely coupled services that can operate independently of each other. Each service can consume and generate events as needed, without having to worry about the implementation details of other services.

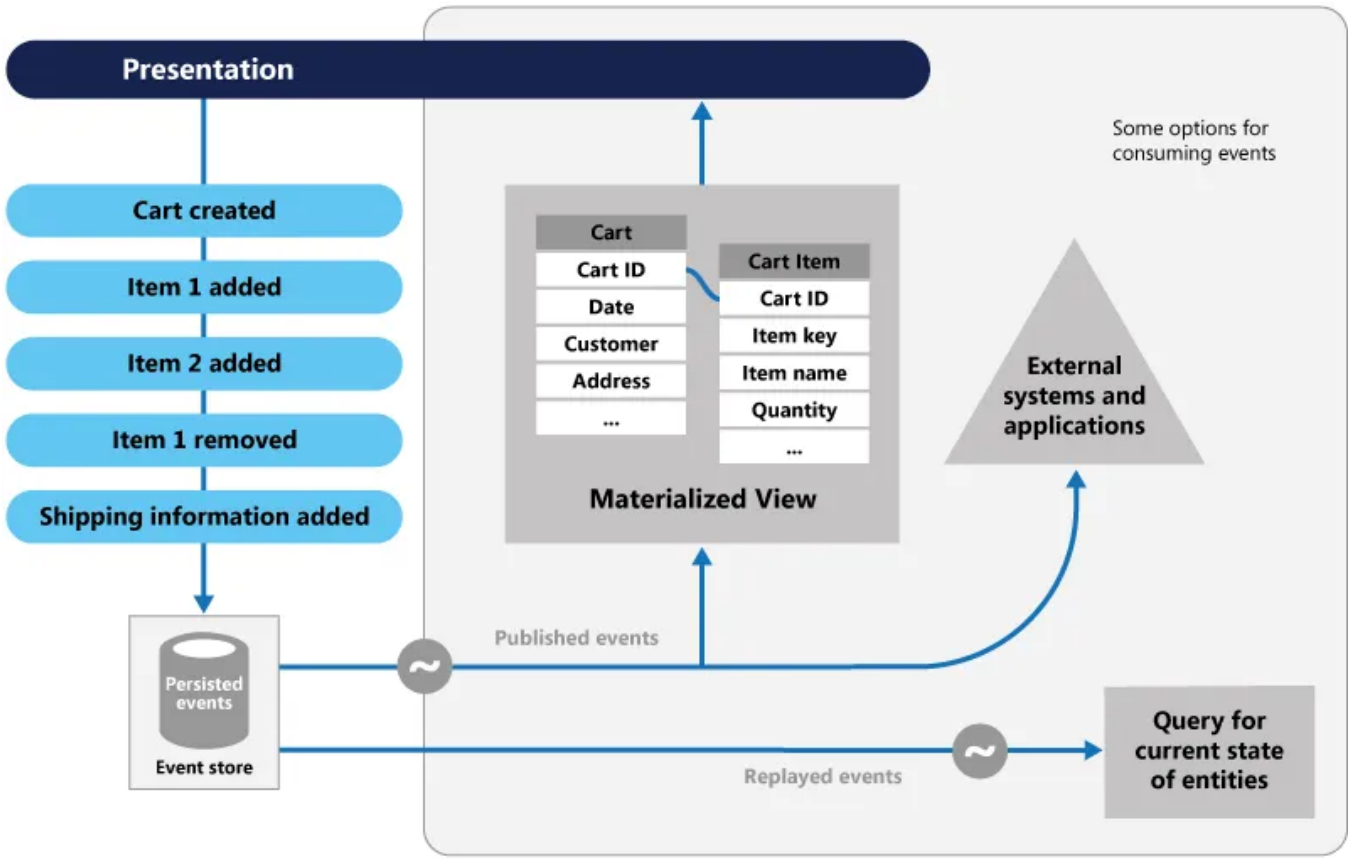
In terms of managing distributed transactions, *Event Sourcing provides a way to ensure that each service involved in a transaction can reliably commit or rollback its changes.*

When a transaction involves multiple services, **each service can publish an event indicating whether it has successfully completed its part of the transaction or not.**

Other services can then consume these events and act accordingly, ensuring that the transaction is either committed or rolled back consistently across all services involved.

Overall, Event Sourcing is a powerful pattern for managing distributed transactions in Microservices architecture, providing a scalable and reliable way to handle complex business processes.

Here is a nice diagram which explains Event Sourcing pattern:



How does Spring Cloud help with distributed transaction management

Spring Cloud provides several features that help with distributed transaction management in microservices architecture:

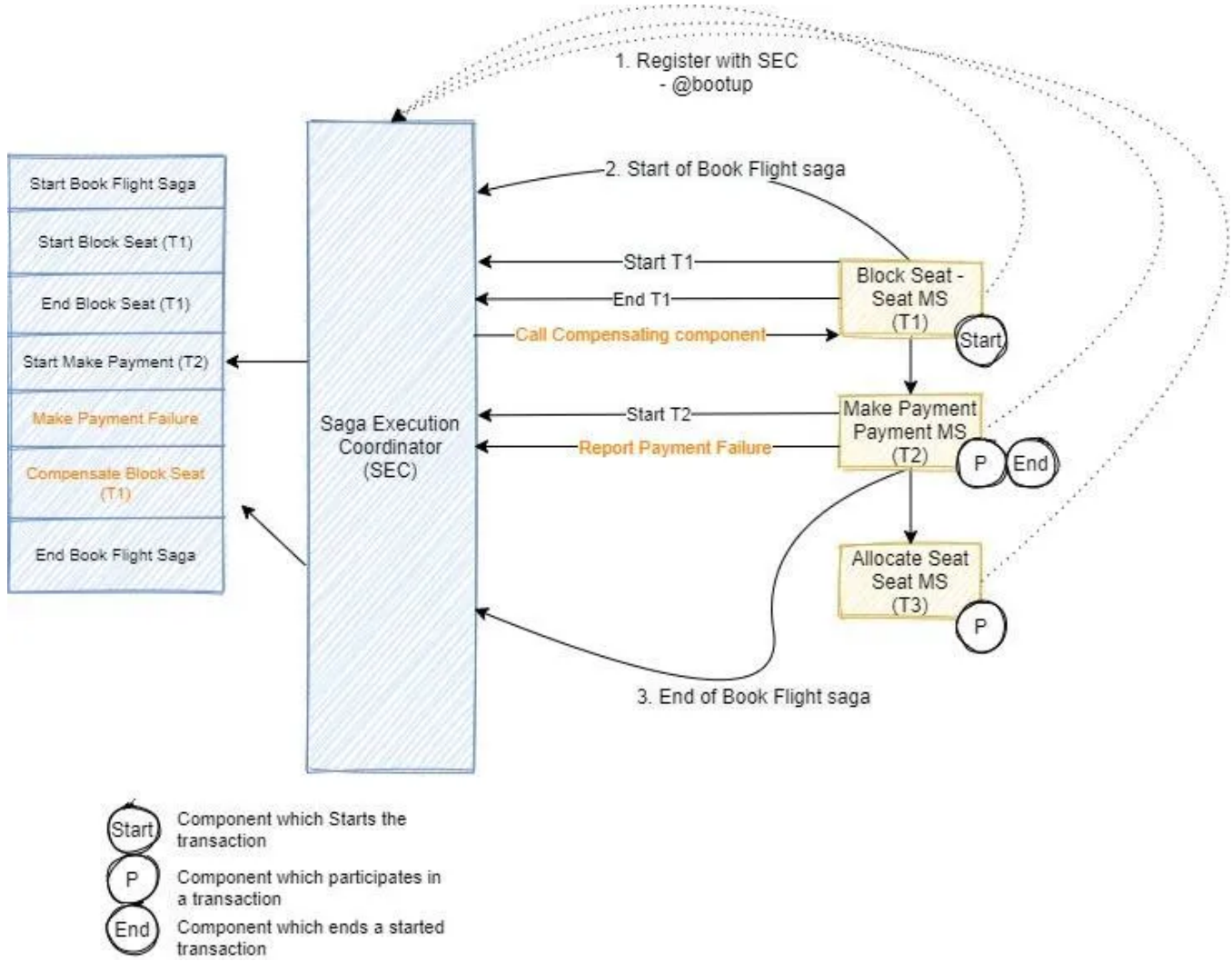
1. **Service Registry and Discovery**
   Spring Cloud's Service Registry and Discovery allow services to discover and communicate with each other easily. This helps with managing distributed transactions by providing a central location for service discovery and registration.

2. **Circuit Breaker**
   Spring Cloud's Circuit Breaker pattern helps to prevent cascading failures in distributed systems. By isolating failures in a specific service, the Circuit Breaker pattern prevents the failure from affecting other services in the system and maintains system availability.

3. **Distributed Tracing**
   Spring Cloud's Distributed Tracing feature helps to track transactions across multiple microservices. This helps to identify the source of issues in a transaction and debug problems easily.

4. **Configuration Management**
   Spring Cloud provides a centralized configuration management system that helps to manage configuration across multiple microservices. This allows the system to be updated and scaled easily, which can help with transaction management in distributed systems.

Overall, Spring Cloud provides a range of tools and features that help to manage distributed transactions in microservices architecture, making it easier to maintain the consistency and reliability of the system



Java and Spring Interview Preparation Material

Before any Java and Spring Developer interview, I always use to read the below resources

Grokking the Java Interview

Grokking the Java Interview: click here

I have personally bought these books to speed up my preparation.

You can get your sample copy here, check the content of it and go for it

Grokking the Java Interview [Free Sample Copy]: click here

> *If you want to prepare for the Spring Boot interview you follow this consolidated ebook, it also contains microservice questions from spring boot interviews.*

Grokking the Spring Boot Interview

You can get your copy here — **Grokking the Spring Boot Interview**

Conclusion

That's all about how to manage transaction in distributed system. In conclusion, transaction management in microservices and distributed systems can be a challenging task, is not easy but it is crucial for ensuring data integrity and consistency across the system.

It is essential to understand the various transaction management patterns, such as two-phase commit, SAGA, and event sourcing, and select the appropriate approach for your system's specific requirements. Each pattern has its benefits and drawbacks, and the selection should be based on the system's requirements and constraints.

While **two-phase commit provides consistency**, it has **a single point of failure, and the performance can be impacted.** On the other hand, **SAGA and event sourcing provide better scalability** and availability but require additional efforts for implementation and maintenance.

Therefore, *it is really important for a developer and architect to carefully analyze and design the transaction management system for your microservices and distributed system to ensure that the system performs optimally*, and data integrity is maintained.

As microservices and distributed systems continue to gain popularity, the transaction management challenge will continue to evolve, and new patterns and techniques will emerge. Keeping up with the latest developments in this space will be crucial for developing efficient and robust microservices and distributed systems.

And , if you are not a Medium member then I highly recommend you to join Medium and read great stories from great authors from real field. You can **join Medium here**

Other Stories you may like

Difference between API Gateway and Load Balancer in Microservices?

Hello folks, what is difference between API Gateway and Load balancer is one of the popular Microservice interview...

medium.com

21 Software Design Pattern Interview Questions and Answers

Design patterns are important topic for Java interviews, here are common design pattern questions you can prepare for...

medium.com

Microservices Programming Cloud Computing Java Software Development

Following

Written by Soma

4.1K Followers · Editor for Javarevisited

Java and React developer, Join Medium (my favorite Java subscription) using my link 👉
https://medium.com/@somasharma_81597/membership

More from Soma and Javarevisited

Soma in Javarevisited

10 Java Features That Make Backend Development Easier

My favorite Java features for back-end or server side app development

· 6 min read · Jul 18

168

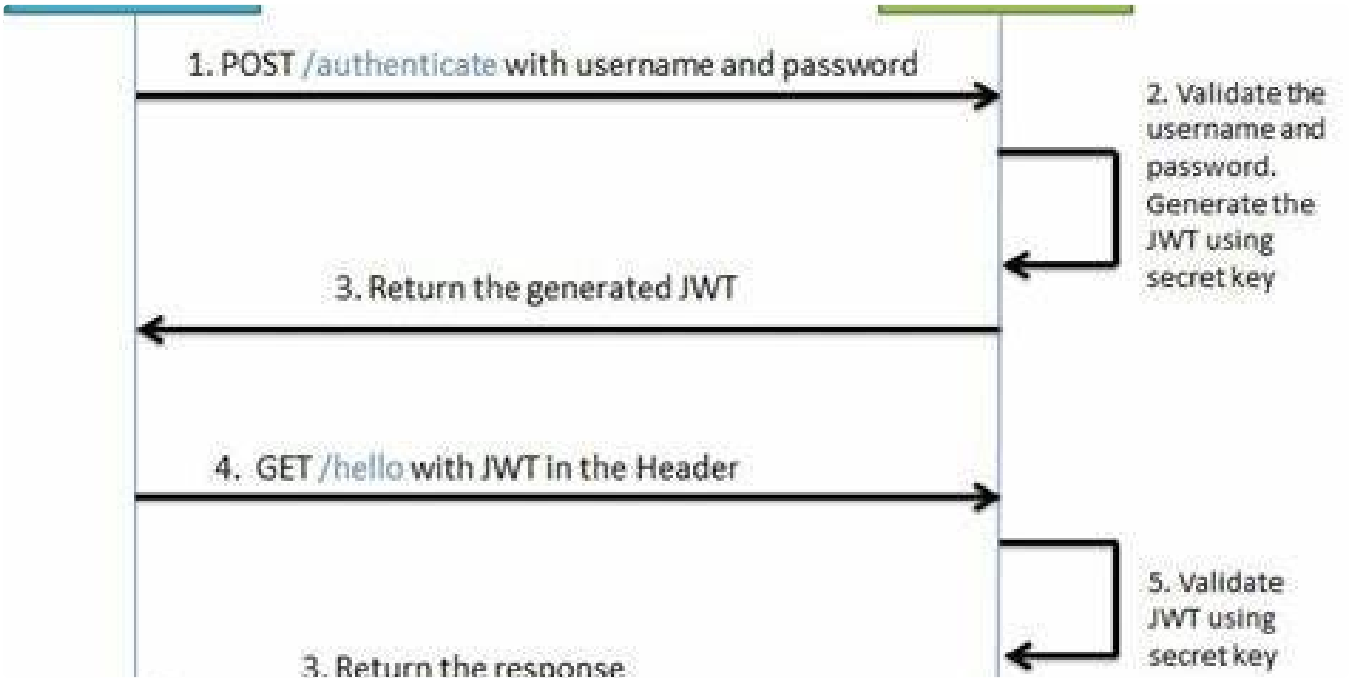
javinpaul in Javarevisited

Top 50 Data Structure and Algorithms Interview Questions for Programmers

Preparing for Programming job interviews? Here are 50+ Data Structure and algorithms problems you can practice to revise key concepts for...

13 min read · Sep 23, 2018

23K 34



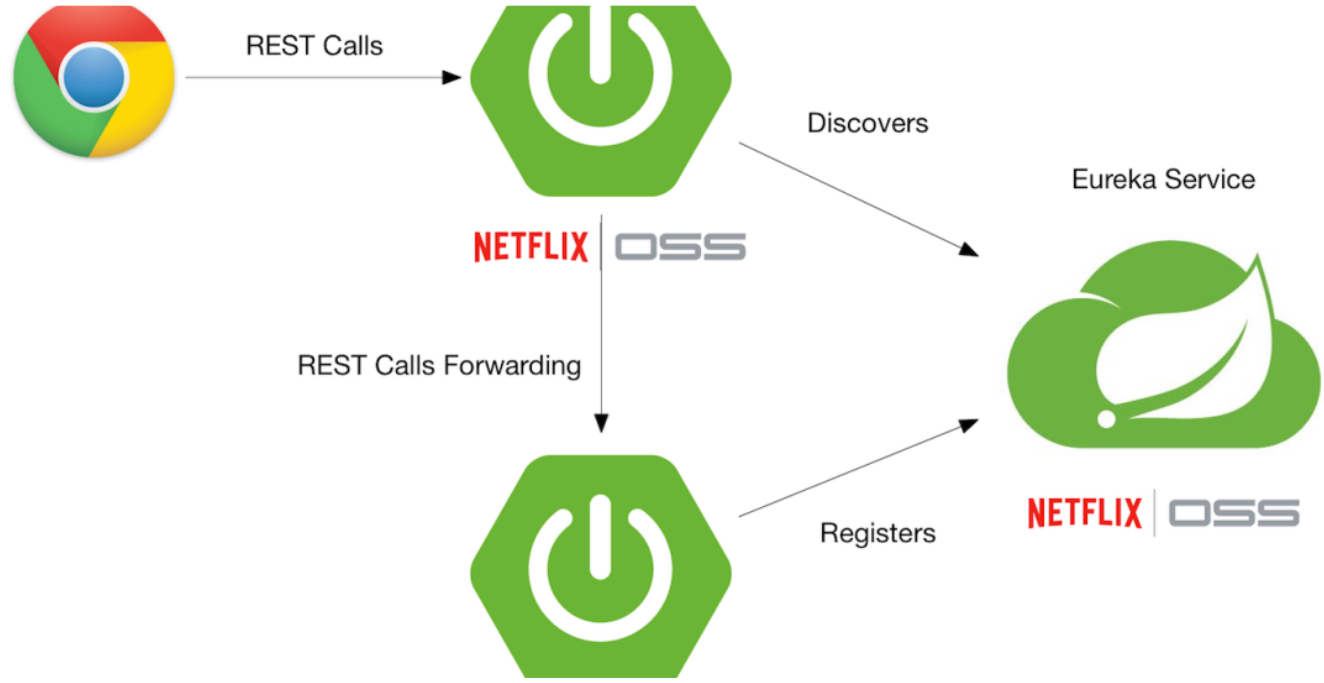
Kishore Karnatakapu in Javarevisited

How to secure Spring Boot with JWT Authentication and Authorization

Hello learners, here we are going to learn about spring security implementation with spring boot3.0 and JWT. This is in continuation with...

4 min read · Apr 19

200 16



Soma in Javarevisited

How to use Spring Cloud Eureka for Service Discovery in Microservices?

Simplifying Microservices Communication with Spring Cloud Eureka: A Comprehensive Guide

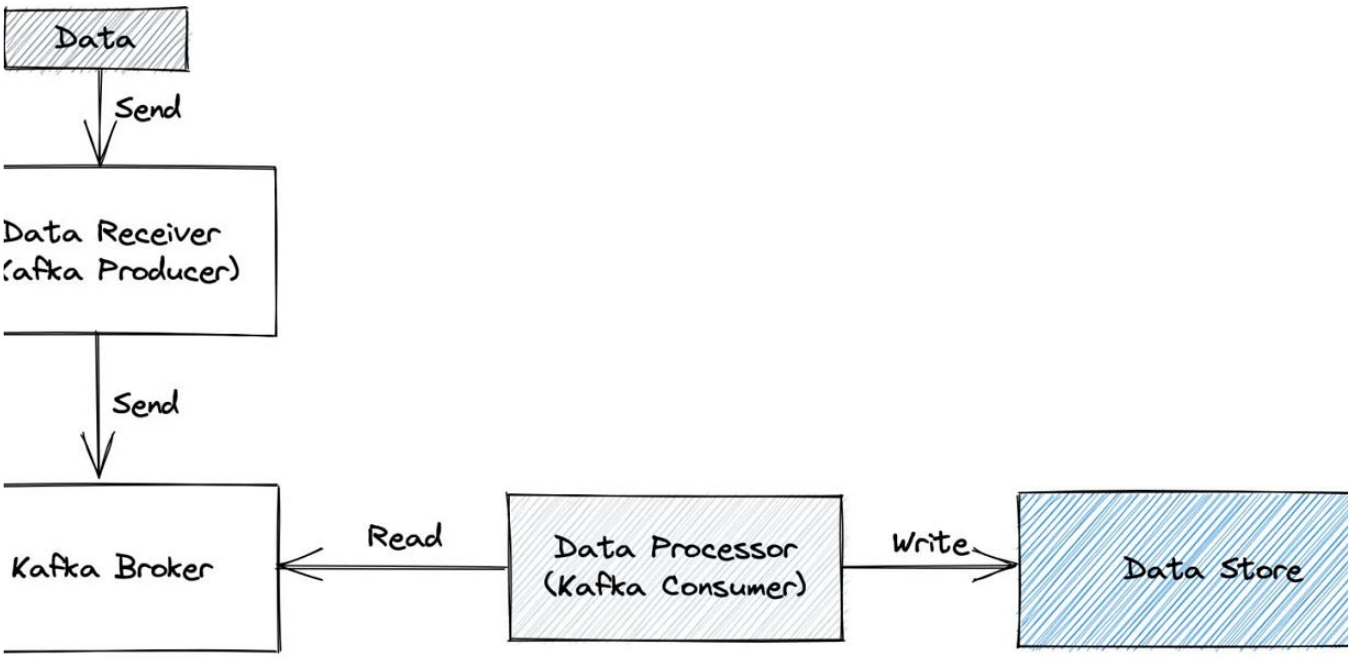
· 9 min read · Jul 25

107

See all from Soma

See all from Javarevisited

Recommended from Medium



Kajal Rawal

Causes & Fix for Consumer Lag in Kafka

What Causes Kafka Consumer Lag?

· 4 min read · Mar 2

8

Vivekanandan Sakthivelu in Gitrebase

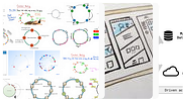
Oh !! Stop using @Builder

The days of writing getters and setters, or generating them using an IDE, are gone. The Lombok plugin has improved developer productivity...

2 min read · Feb 21

595 14

Lists



General Coding Knowledge

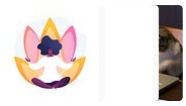
20 stories · 192 saves

It's never too late or early to start something

13 stories · 66 saves

Coding & Development

11 stories · 97 saves



Stories to Help You Grow as a Software Developer

19 stories · 261 saves

Bubu Tripathy

Best Practices for designing REST APIs

In this post, we'll discuss some of the best practices for designing RESTful APIs, including real-world examples.

15 min read · Mar 6

338 9

Ashish Pandey in Excited Developers

Java Multithreading — All you need to know

The only blog that you will ever need. Let's get started

13 min read · Jun 1

69

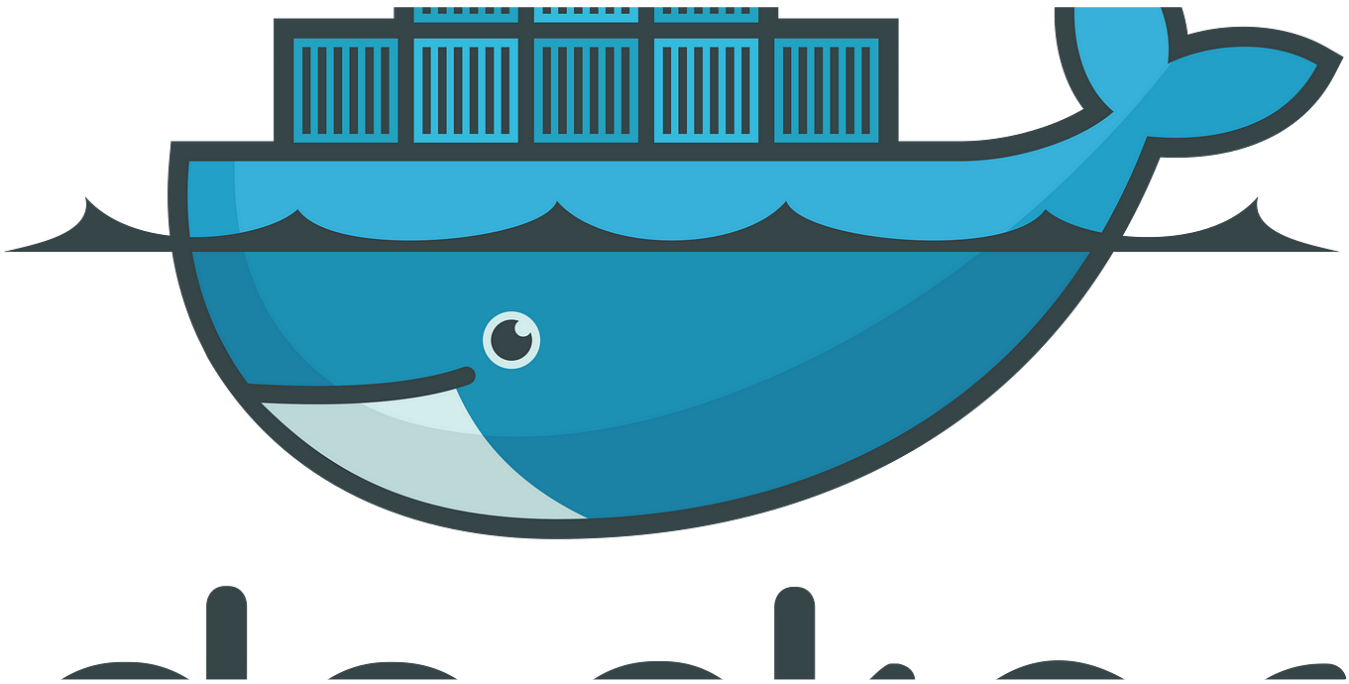
Daryl Goh

Spring Boot: RequestEntity vs ResponseEntity | RequestBody vs ResponseBody

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.

· 5 min read · May 21

40



EazyPeazyGeeky

How to Install and Use Docker on Ubuntu 22.04

Docker is a platform that allows you to easily create, deploy, and run applications in containers. Containers are lightweight, portable...

2 min read · Jul 30