



★ Member-only story

Difference between Synchronous vs Asynchronous Communication in Microservices

Choosing the Right Communication Pattern for Microservices Architecture:
Exploring Synchronous and Asynchronous Communication Methods



Soma · [Follow](#)

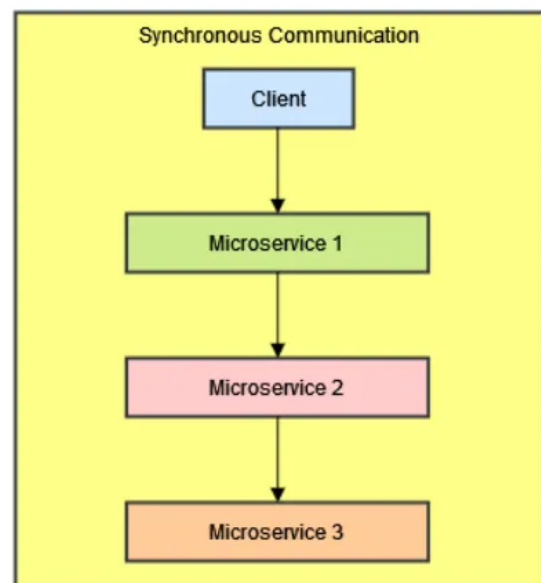
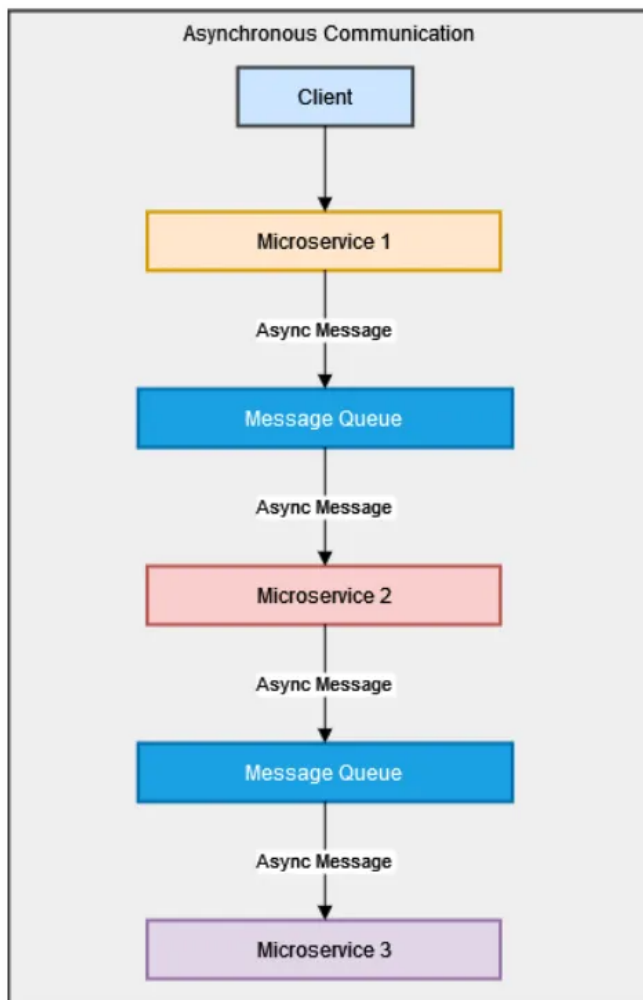
Published in Javarevisited

10 min read · Apr 11

Listen

Share

More



Hello folks, if you are working on **Microservice architecture** and wondering *how does Microservices communicate with each other* and *what is difference between synchronous and asynchronous communication* and *when to use one over other?* then you have come to the right place.

In the last few articles I have talked about common Microservice design patterns like **Event Sourcing**, **CQRS**, **SAGA**, **Database Per Microservices**, **API Gateway**, **Circuit-Breaker** and also shared **best practices to design Microservices** and now, you will learn about Microservice communication, including synchronous and asynchronous communication using Message queue like RabbitMQ and Apache Kafka.

In recent years, **Microservices architecture** has gained popularity as an approach to building scalable and modular applications. One of the key considerations in microservices design is how microservices communicate with each other. In this article, we will explore two common methods of communication between microservices: **synchronous and asynchronous communication**.

We will discuss the *characteristics, advantages, and disadvantages of each approach*, along with real-world examples of when to use synchronous or asynchronous communication in microservices architecture.

Understanding the nuances of these **communication patterns** is crucial in building efficient and resilient Microservices-based systems. So let's dive into the world of microservices communication and explore the differences between synchronous and asynchronous communication in this comprehensive article.

But, 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**

Join Medium with my referral link - Soma

Read this and every story from thousands of great writers on Medium). Read more from great writers on Medium like...

medium.com

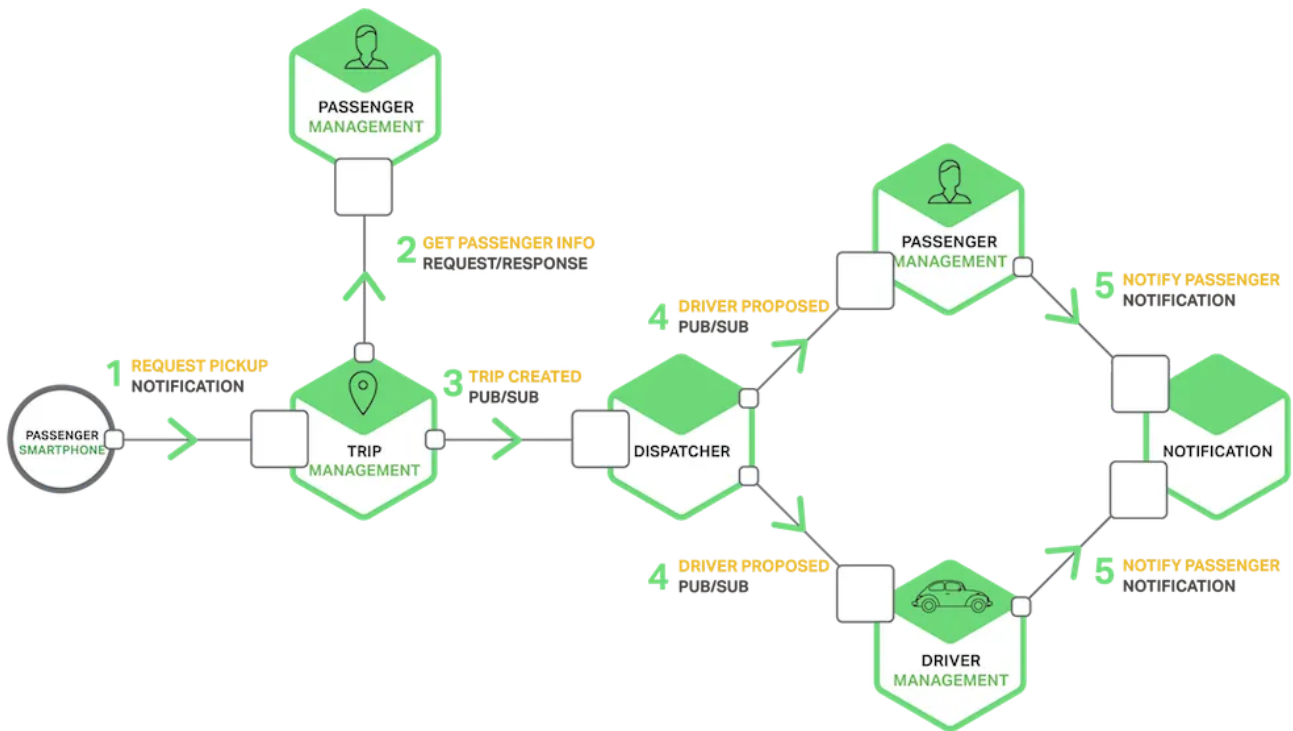
Why Microservices communication is important?

If you have noticed, you never develop your application as single Microservices, but a collection of Microservices. After all, you are breaking Monolith to multiple *Microservices*, hence, Microservices communication is crucial in a microservices architecture because it enables individual microservices to work together seamlessly as a cohesive system. It's also one of the *10 things you should consider while designing and developing your Microservice App.*

Microservices are designed to be loosely coupled and independently deployable, which means they need to communicate effectively to share data, coordinate actions, and collaborate in order to fulfill the overall business requirements.

Efficient communication among microservices is also essential for maintaining **system integrity, scalability, and reliability**. Proper communication patterns also play a crucial role in enabling fault tolerance, resilience, and graceful degradation in the face of failures or changes in the system.

That's why **choosing the right communication methods is therefore crucial for building and maintaining a successful Microservice application**. Now that you know the importance of communication, let's explore synchronous and asynchronous communication in Microservices.



What is Synchronous communication of Microservices?

Synchronous communication in Microservices refers to a communication pattern where the **client making a request to a microservice waits for a response** before proceeding with further actions.

In other words, the client blocks and waits for a response from the microservice before continuing its execution.

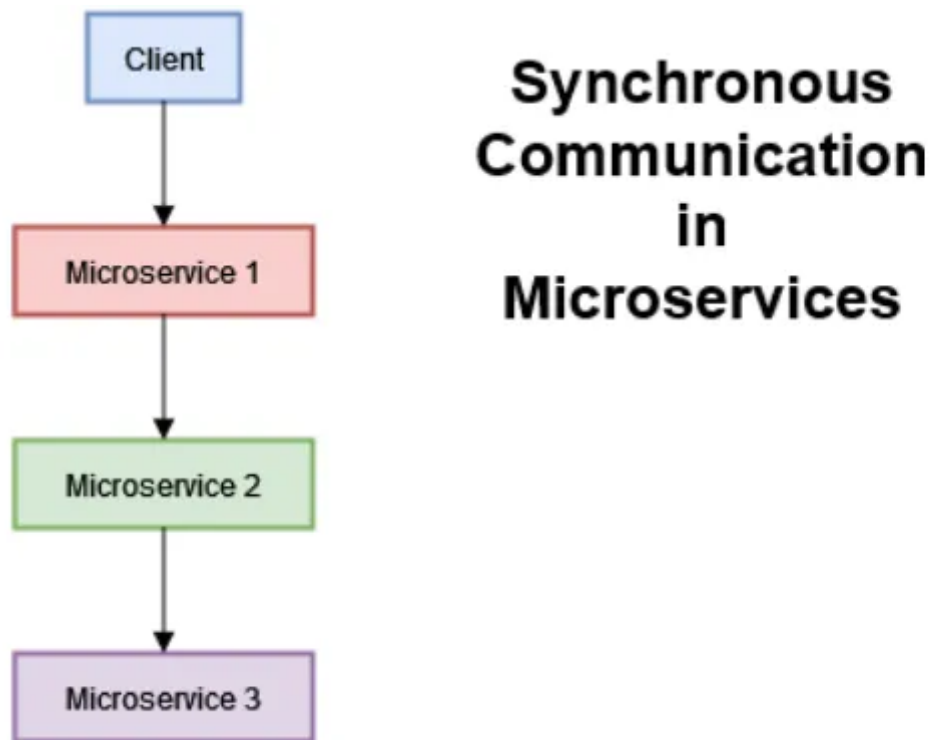
This communication pattern is typically **achieved using HTTP-based protocols such as REST (Representational State Transfer) or SOAP (Simple Object Access Protocol)**, where the client sends a request to the microservice and waits for a response.

In synchronous communication, the client and microservice **are tightly coupled**, as the client has to wait for the microservice to complete its processing before it can continue.

This can result in **potential performance and scalability issues**, as it may lead to *increased response times and reduced system throughput*, especially in scenarios where microservices have high processing times or high concurrency.

However, synchronous communication *can be simpler to implement and may be suitable for use cases where immediate responses are required* or where the microservice needs to perform a task before responding to the client. It can also be useful in cases where strong consistency and coordination are necessary between microservices, such as in transactional scenarios.

Here is an **example of Synchronous Communication in Microservices**



In this example, the client sends a request to Microservice 1 (Order Microservice), which in turn communicates with Microservice 2 (Inventory Microservice), and so on. The client waits for a response from each microservice before proceeding to the next one.

What is Asynchronous Communication in Microservices? how does it achieved?

Asynchronous communication in Microservices refers to a communication pattern where the client making a request to a microservice **does not wait** for an immediate response.

Instead, **the client continues its execution without blocking**, and the microservice processes the request independently. The response is usually sent later, either as a callback or through a separate channel.

Asynchronous communication can be achieved through various mechanisms, such as **message queues, event-driven architectures, and publish-subscribe patterns**.

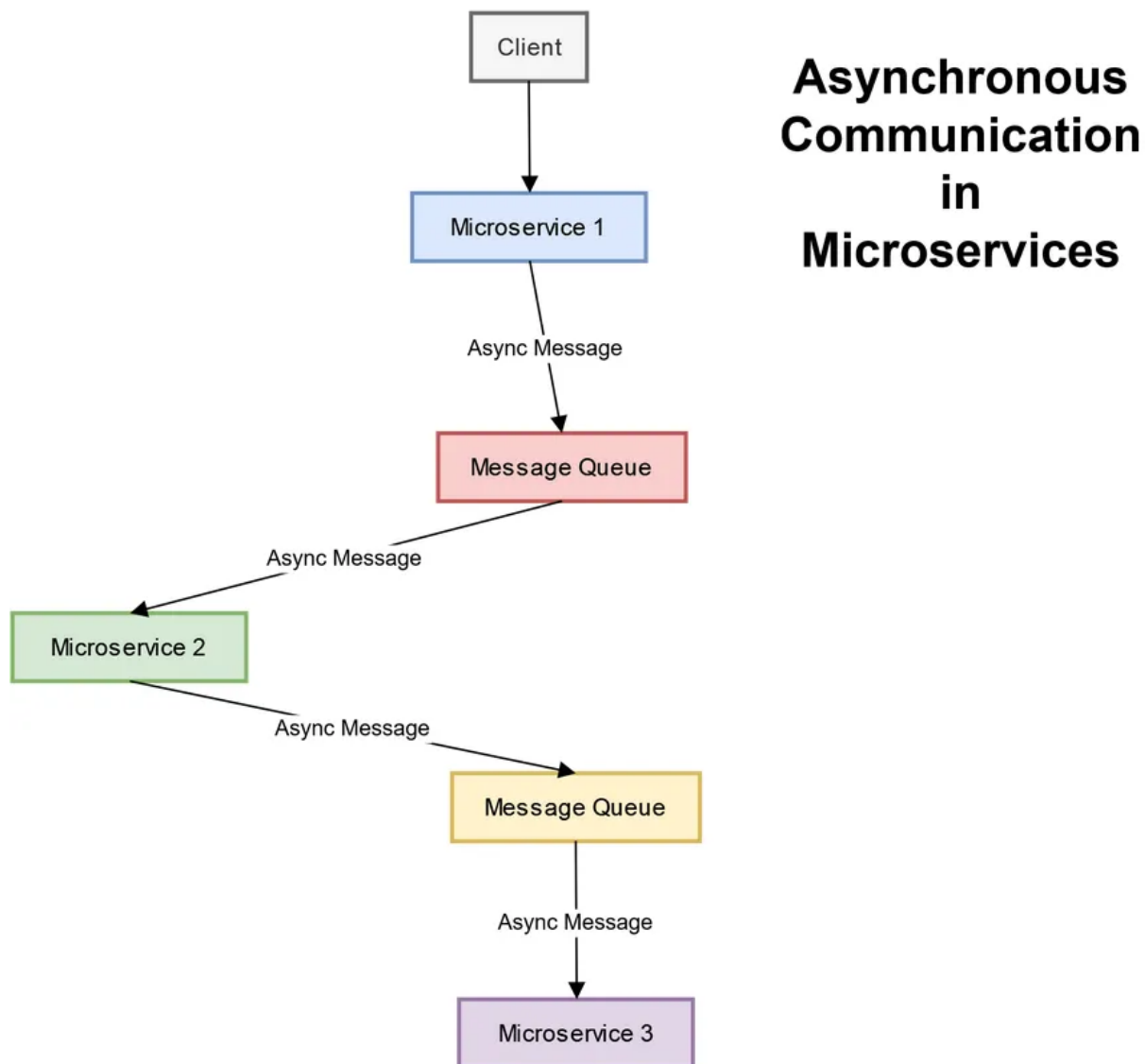
Microservices can *send messages* or events to message brokers or event buses, which then deliver them to the appropriate microservices that have registered to receive those messages or events. This **decouples the sender and receiver microservices**, allowing them to evolve independently and operate at different speeds.

Asynchronous communication also provides several benefits in microservices architectures. It can **improve system scalability, fault tolerance, and responsiveness**.

Microservices can also process requests in parallel, and failures in one microservice do not necessarily impact the immediate response to the client.

It also **enables loose coupling between microservices**, allowing them to evolve independently and be more resilient to changes in the system. However, *it can also introduce complexities such as eventual consistency, message ordering, and error handling*, which need to be carefully managed.

Here is an example of **Asynchronous Communication in Microservice architecture using Message Queues**



In this example, the client sends a message to Microservice 1, which **puts the message in a message queue**. Microservice 2 then consumes the message from the queue and processes it asynchronously.

This pattern allows microservices to communicate independently and asynchronously, without blocking the client or each other.

Difference between synchronous and asynchronous communication Patten in Microservices

Now that you have understood what is Synchronous and Asynchronous communication in Microservice architecture, you can easily answer this question. But for your convenience, here are the key differences between synchronous and asynchronous communication patterns in microservices are:

1. Request-Response vs. Fire-and-Forget

In synchronous communication, the client making a request to a microservice **waits** for an immediate response before proceeding, while in asynchronous communication, the client *does not wait* for a response and continues its execution.

2. Blocking vs. Non-blocking:

Synchronous communication *blocks the client* until a response is received, whereas asynchronous communication allows the client to continue its execution *without blocking*, as the microservice processes the request independently.

3. Immediate vs. Delayed Response

Synchronous communication provides *an immediate response*, while asynchronous communication may have a delayed response, as the microservice processes the request and sends the response later, either as a *callback* or through a separate channel.

4. Coupling vs. Decoupling

Synchronous communication can introduce *tight coupling* between microservices, as they need to be available and respond immediately to each other's requests. In contrast, asynchronous communication allows for *loose coupling*, as microservices can operate independently and asynchronously process requests.

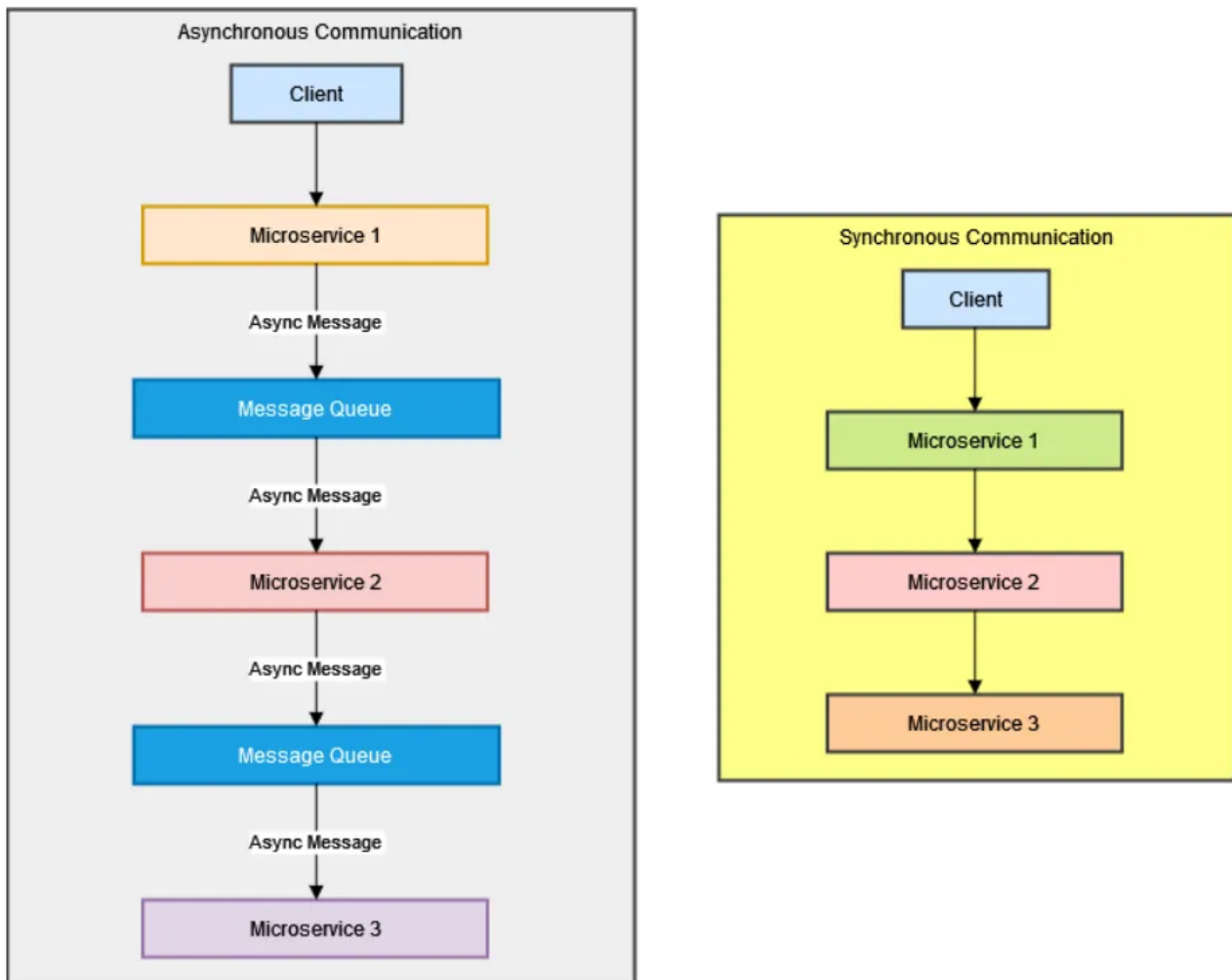
5. Scalability and Fault Tolerance

Asynchronous communication can *improve system scalability*, as microservices can process requests in parallel, and failures in one microservice do not necessarily impact the immediate response to the client. Synchronous communication, on the other hand, may require additional resources to handle multiple concurrent requests and may be more prone to cascading failures.

6. Complexity

Asynchronous communication can *introduce complexities such as eventual consistency, message ordering, and error handling*, as messages or events may be processed out of order or lost. Synchronous communication, on the other hand, is relatively simpler as it provides immediate responses and does not require additional mechanisms for message handling.

Overall, *synchronous communication is suitable for cases where immediate responses are critical*, and microservices need to be tightly coupled, while asynchronous communication is beneficial for cases where decoupling, scalability, and fault tolerance are important, and delayed responses are acceptable trade-offs.



When to use synchronous vs asynchronous communication in Microservices?

As I said, the decision to use synchronous or asynchronous communication in microservices depends on several factors, including the specific requirements and constraints of the application and the trade-offs that are acceptable in terms of performance, scalability, and fault tolerance.

But, here are some general guidelines which can give you ideas when to use Synchronous and Async communication in Microservice architecture:

Use Synchronous Communication when:

1. Immediate response is critical

If your microservice clients require immediate responses and cannot proceed without them, synchronous communication may be more suitable. For example, when a client needs to retrieve real-time data or perform a transaction that requires immediate confirmation.

2. Tight coupling is acceptable

Synchronous communication typically requires more coupling between microservices, as they need to be available and respond immediately to each other's requests. If tight coupling is acceptable and necessary for the application, synchronous communication may be preferred.

3. Simplicity is a priority

Synchronous communication is relatively simpler as it provides immediate responses and does not require additional mechanisms for message handling or event processing.

If simplicity is a priority and the additional complexities of asynchronous communication are not necessary, synchronous communication may be more straightforward.

Use Asynchronous Communication when:

1. Decoupling is desired

If the microservices need to operate independently and be loosely coupled, asynchronous communication may be more suitable. It allows microservices to communicate without being tightly coupled, enabling them to evolve independently and scale individually.

2. Scalability is important

Asynchronous communication can improve system scalability, as microservices can process requests in parallel, leading to better performance and scalability. If scalability is a key requirement, asynchronous communication may be preferred.

3. Fault tolerance is a priority

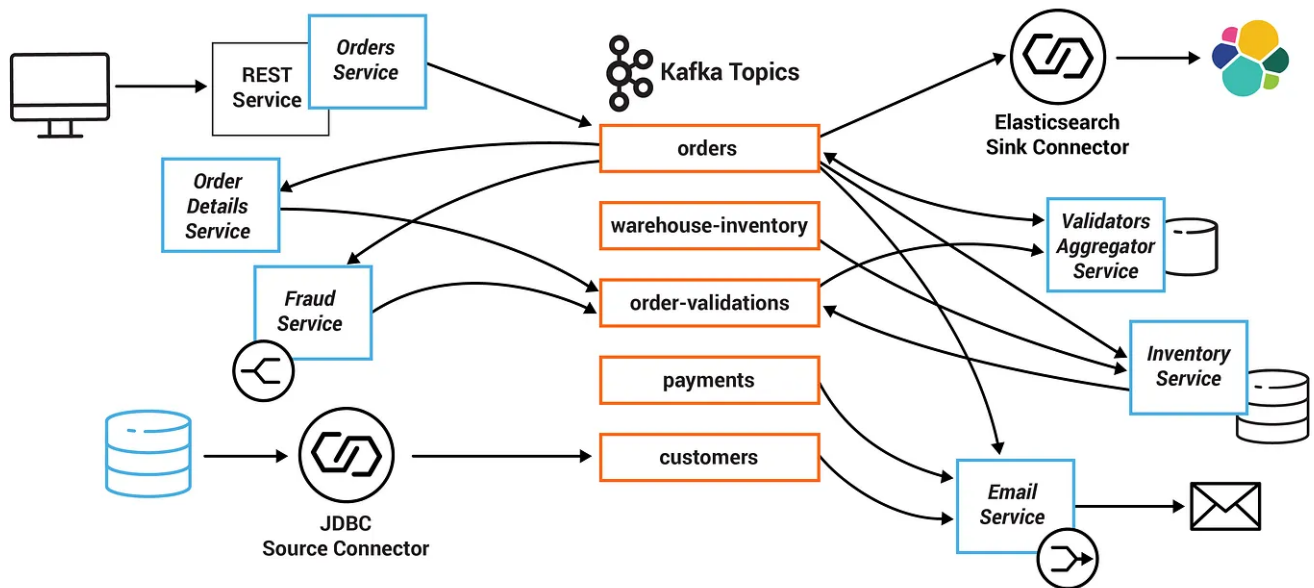
Asynchronous communication can provide better fault tolerance, as failures in one microservice do not necessarily impact the immediate response to the client. If fault tolerance is a priority, asynchronous communication may be more suitable as it can tolerate failures in microservices without impacting the overall system.

4. Delayed response is acceptable

Asynchronous communication may introduce a delay in response time, as microservices process requests and send responses later, either as callbacks or through separate channels. If delayed response time is acceptable for the application, asynchronous communication may be preferred.

In summary, the choice between synchronous and asynchronous communication in microservices depends on the specific requirements, constraints, and trade-offs of the application.

Synchronous communication may be suitable for cases where immediate response, tight coupling, and simplicity are important, while asynchronous communication may be preferred for cases where decoupling, scalability, fault tolerance, and delayed response are prioritized.



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](#)



That's all about how **Microservices communicates with each other in distributed systems**. As I have said, communication between microservices is a crucial aspect of building distributed systems using the microservices architecture.

Synchronous and asynchronous communication are two common patterns used for microservices communication, each with its advantages and trade-offs.

Synchronous communication provides immediate responses and is relatively simpler, but can lead to **tight coupling** and may not be suitable for highly scalable and fault-tolerant systems.

On the other hand, *asynchronous communication enables loose coupling, better scalability, and fault tolerance*, but may introduce delays in response time and require additional mechanisms for message handling.

The choice between synchronous and asynchronous communication depends on the specific requirements and constraints of the application. Factors such as the need for immediate response, tight coupling, simplicity, decoupling, scalability, fault tolerance, and acceptable response time delay should be considered when deciding which communication pattern to use.

This is also an important topic for Microservice interview point of view and by understanding the pros and cons of synchronous and asynchronous communication you can make informed decisions to effectively design and implement microservices-based systems that meet the desired performance, scalability, and fault tolerance requirements.

If you like this Microservice article and want to read more such article but you are not a Medium member then I highly recommend you to join Medium and many such articles from many Java and Microservice experts. You can **join Medium [here](#)**

Join Medium with my referral link - Soma

Read this and every story from thousands of great writers on Medium). Read more from great writers on Medium like...

medium.com

Other Microservices Articles 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

Top 10 Microservices Problem Solving Questions for 5 to 10 years experienced Developers

From Service Discovery to Security: 10 Problem-Solving Questions to Test the Microservices Skills of Developers with 5...

medium.com

Securing Microservices with OAuth2 and Spring Security

An Overview of Implementing OAuth2 and Spring Security to Enhance Microservices Security

medium.com

Difference between @RequestParam and @PathVariable in Spring MVC?

How to choose between the @RequestParam and @PathVariable in Spring MVC

medium.com

Microservices

Programming

Spring Boot

Java

Software Development



Follow

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

50+ Spring Boot Interview Questions and Answers for Java Programmers

These are 50+ Spring Boot Interview Questions and answers to Crack your Next Java Developer interview

🌟 · 19 min read · Apr 24

 182





 javinpaul in Javarevisited

10 Best Backend Frameworks for Web Development in 2023

These are the best backend development frameworks for web development in Java, Ruby, Python, JavaScript, PHP, Scala, and Golang in 2023

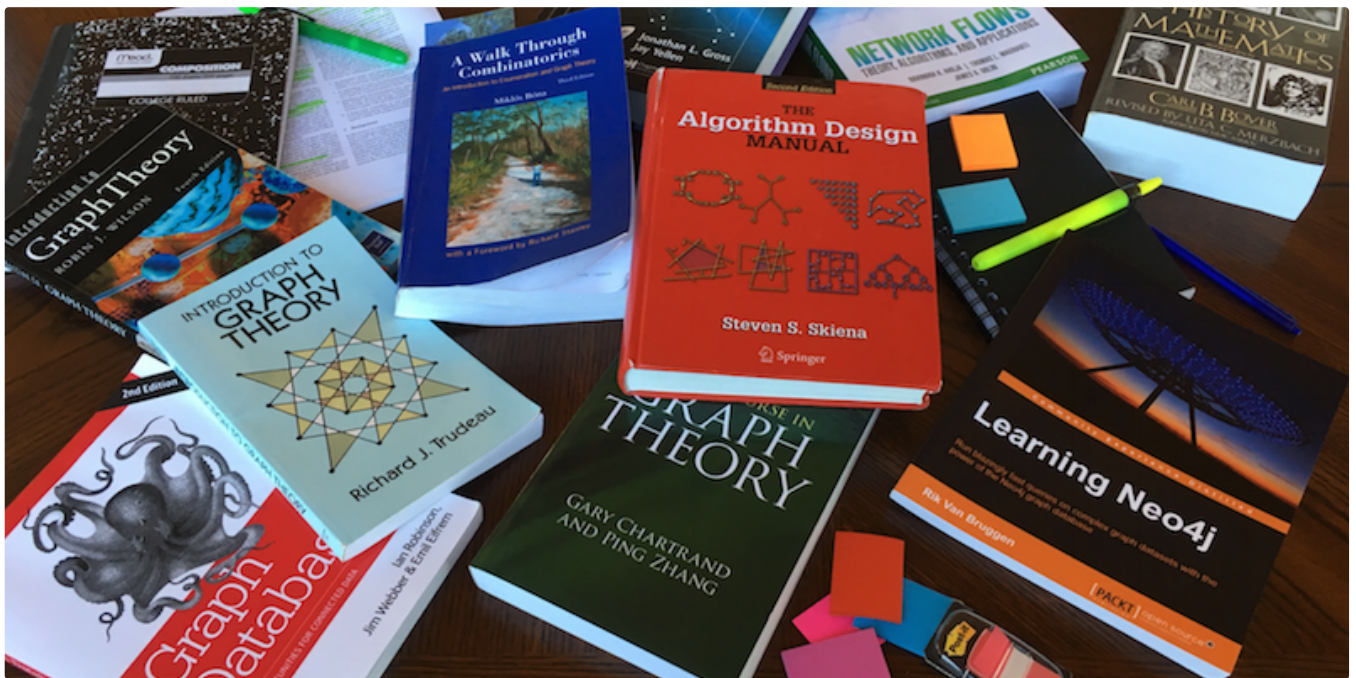
9 min read · Mar 9, 2022




398



5



 javinpaul in Javarevisited

10 Best Books for Data Structure and Algorithms for Beginners in Java, C/C++, and Python

Algorithms are language agnostic, and any programmer worth their salt should be able to convert them to code in their programming language...

12 min read · Mar 13, 2020

 1.2K

 1





Messaging Model	Traditional	Publish/Subscribe	Traditional
Scalability	Clustering/Network of Brokers	Partitioning	Clustering/Network of Brokers
Performance	Moderate	High	High
Data Persistence	On Disk (default), In-memory	On Disk	On Disk (default), Database
Integration	Programming Languages, Databases, Web Servers	Data Processing Systems, Databases, Data Sources	JMS Clients, Apache Camel, Apache CXF
Suitable For	Strict Ordering, Reliable Delivery, Moderate-High	Streaming Data, High Message Rates	High Data Durability, High Performance

 Soma in Javarevisited

Difference between RabbitMQ, Apache Kafka, and ActiveMQ

Hello folks, if you are preparing for Java Developer interviews the along with Spring Boot, and Microservices, you should also prepare...

★ · 8 min read · May 19

 208

 4






See all from Soma

See all from Javarevisited

Recommended from Medium



 Amit Himani

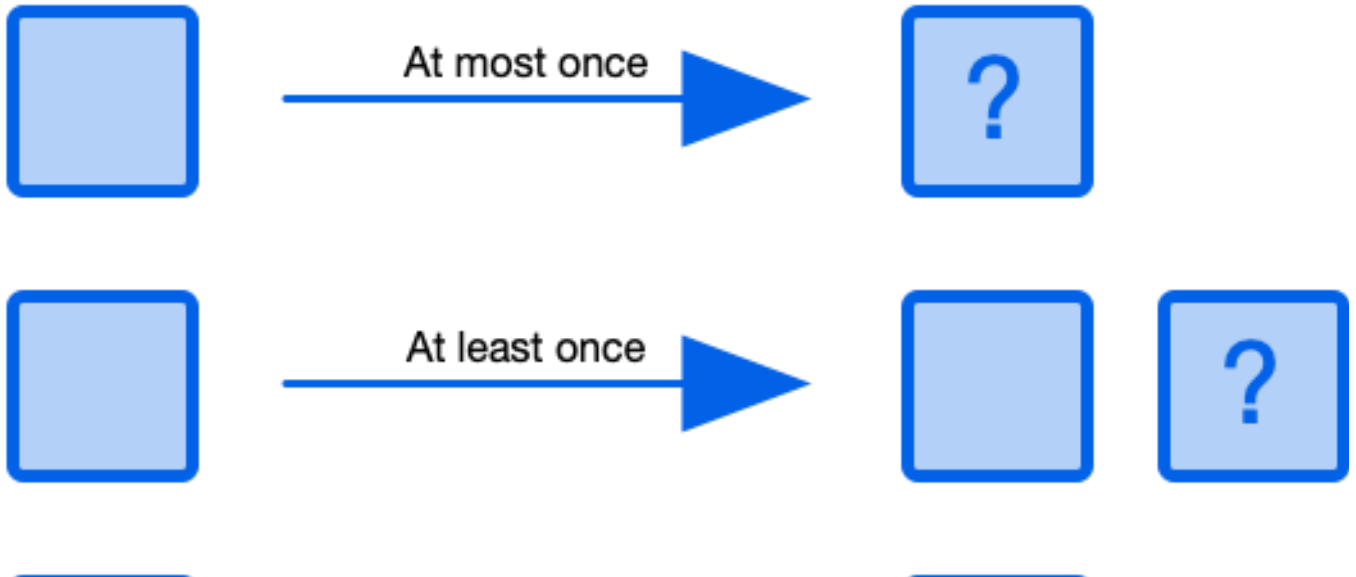
Distributed Transaction in Spring Boot Microservices


Distributed transactions in microservices refer to transactions that involve multiple microservices, each handling a part of the...

★ · 5 min read · Feb 17

 49  3



 Andy Bryant

Processing guarantees in Kafka

Each of the projects I've worked on in the last few years has involved a distributed message system such as AWS SQS, AWS Kinesis and more...

21 min read · Nov 16, 2019

 1.91K

 5



Lists



General Coding Knowledge

20 stories · 200 saves



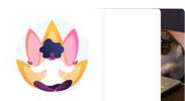
It's never too late or early to start something

13 stories · 68 saves



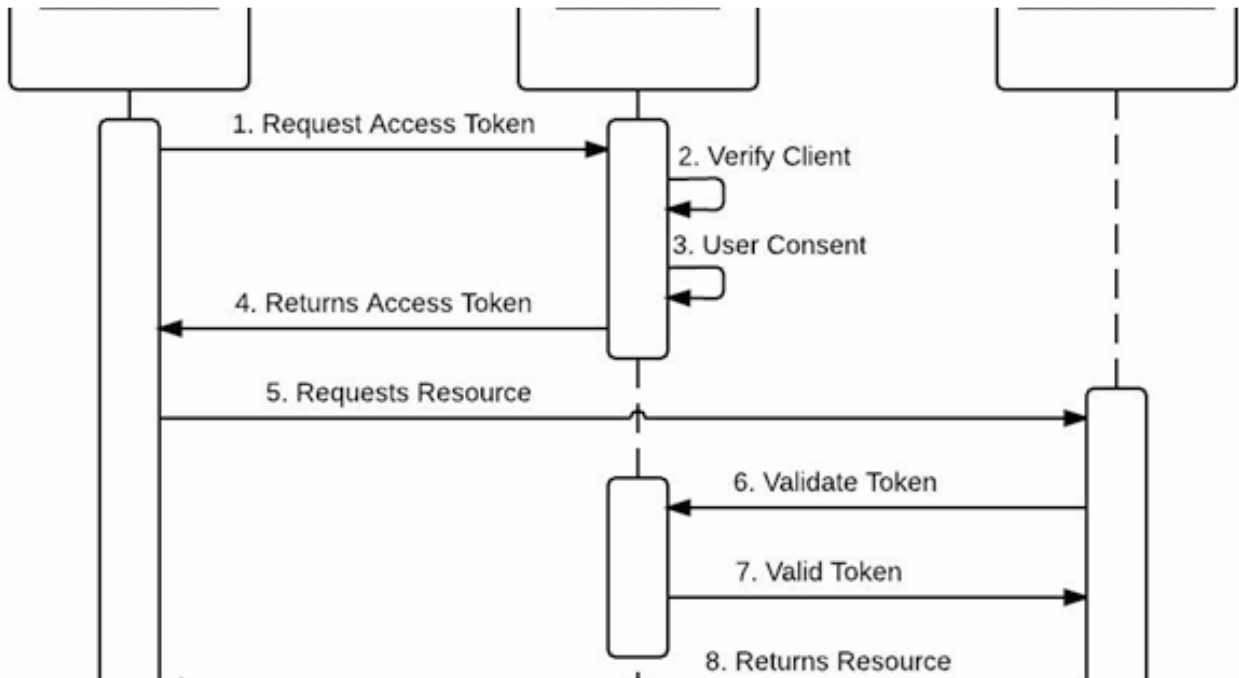
Coding & Development

11 stories · 101 saves



Stories to Help You Grow as a Software Developer

19 stories · 268 saves



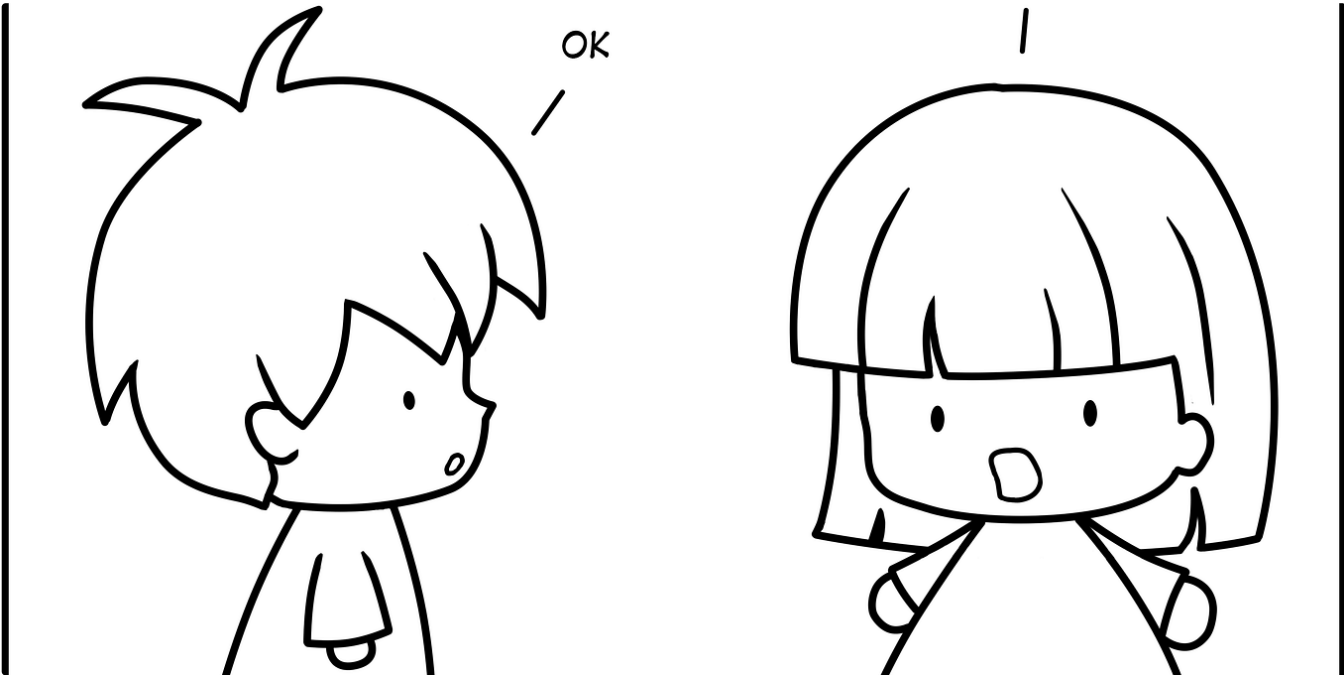
 Soma Chakraborty in Dev Genius


Securing a Microservice with OAuth 2.0 (Part 2 : How Token Based Security Works Internally)

Table of content

5 min read · May 1

 30  1  



 Julie Zhuo in The Year of the Looking Glass

Average Manager vs. Great Manager

Explained in 10 sketches

2 min read · Aug 12, 2015



20K



185



Eray Araz

Spring Webhook

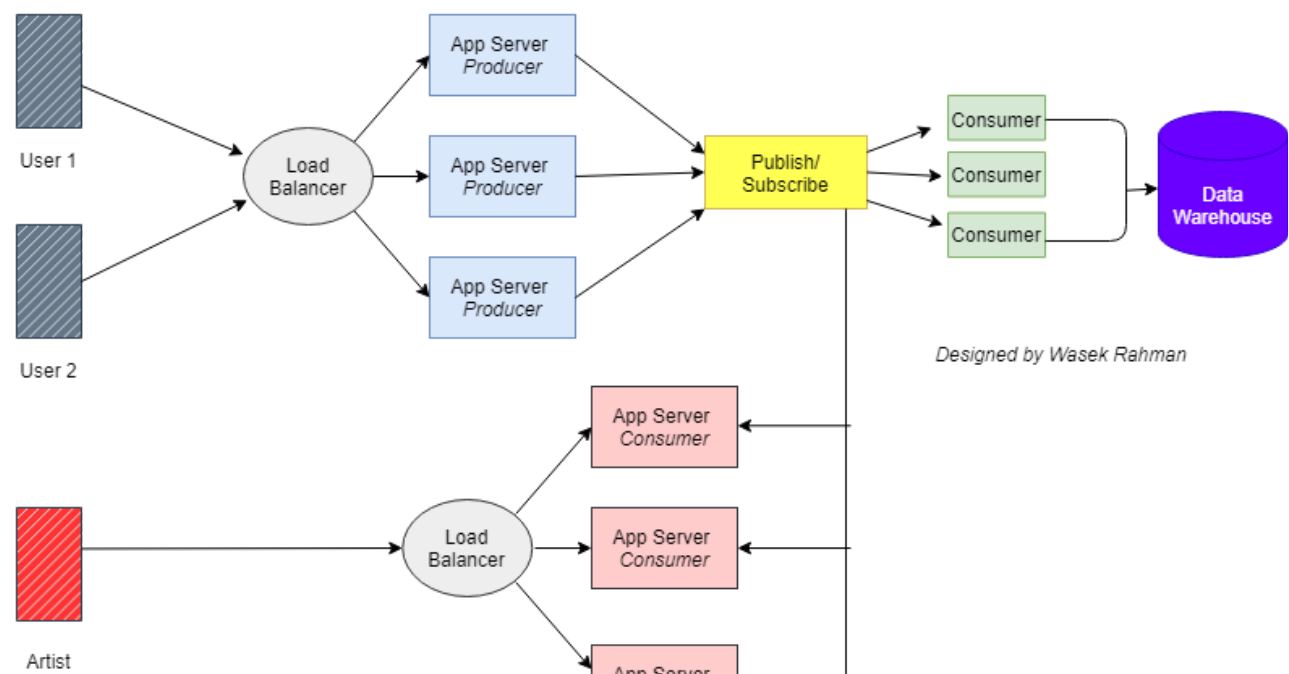
Introduction


3 min read · Apr 2



106





 Wasek Rahman in Bootcamp

System Design Interview Prep: Spotify Real-Time Player

One of the most critical sections of interviewing at a top company or senior role is the System Design section where you would have to come...

🌟 · 4 min read · Feb 18

 120

 1





See more recommendations