

[Open in app](#)

★ Member-only story

# What is Database Per Microservices Pattern? What Problem does it solve?

Breaking Down Monoliths: How the Database Per Microservice Pattern Can Transform Your Architecture

Soma · [Follow](#)

Published in Javarevisited

8 min read · Apr 5



Listen

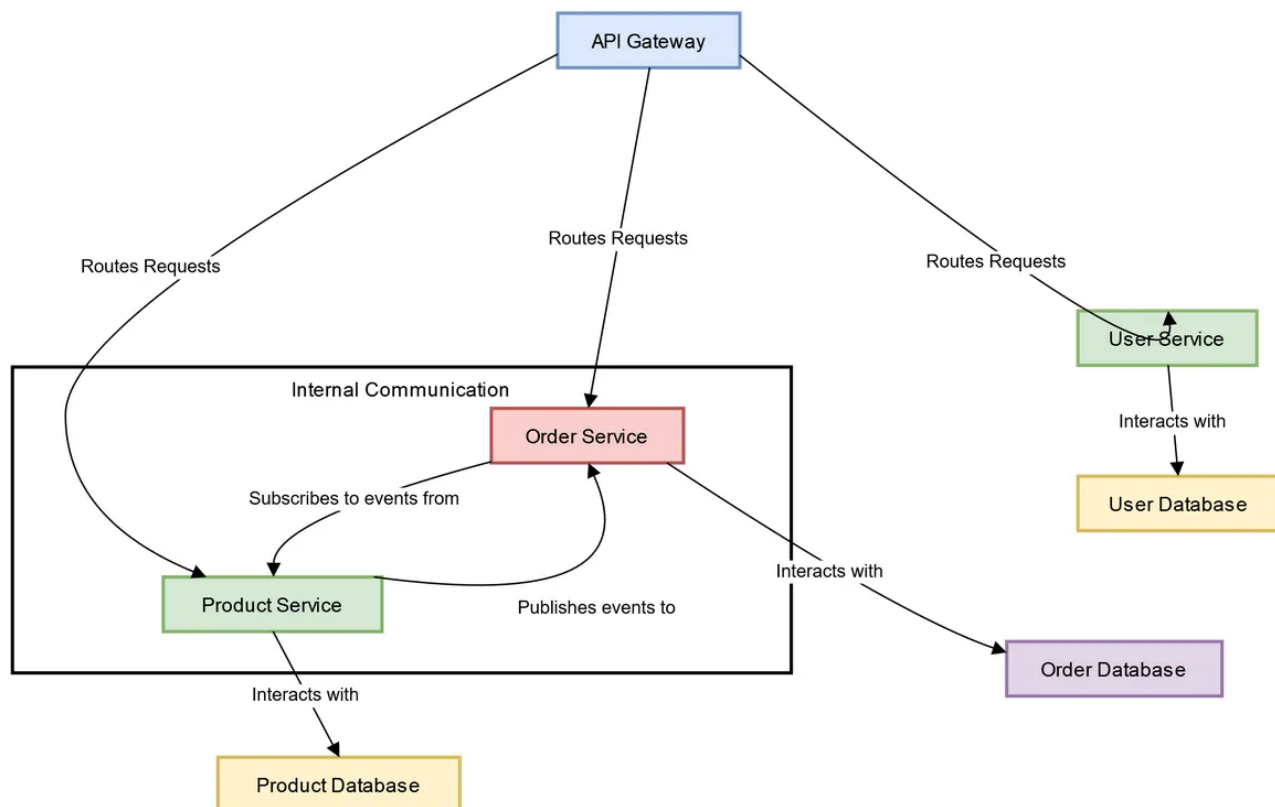


Share



More

## Database Per Microservices Pattern



Hello folks, as the demand for microservices architecture continues to grow mainly due to increased adoption of cloud, developers and architects are constantly seeking ways to optimize their implementation. One approach gaining popularity is the **Database per Microservice design pattern**.

*As the name suggests, this pattern recommends that instead of sharing a common database with other microservices, each microservice should have its own database.*

This approach has several benefits, including **better isolation and scalability, reduced complexity and coupling, improved performance, and easier maintenance**. However, implementing this pattern requires careful consideration of the data access and management strategies, such as *data synchronization and consistency*, database schema design, and deployment strategies.

In the last few articles I have explained popular Microservice design patterns like Event Sourcing, SAGA, API Gateway, Circuit-Breaker, and CQRS and also shared *best practices to design Microservices* and In this article, we will explore the Database per Microservice design pattern and its implementation in different scenarios.

We will discuss the advantages and disadvantages of this pattern, and how it fits with other microservice architecture patterns.

Additionally, we will provide best practices and practical tips for designing and deploying microservices with separate databases, using popular tools and frameworks like Spring Boot, Docker, and Kubernetes, which is also quite important from Microservice interview perspective.

Whether you are a Java Developer, a Microservice architecture enthusiast, a system design expert, or a senior developer seeking to optimize your microservices implementation, this article will provide you with valuable insights and practical guidance on adopting the **Database per Microservice design pattern**.

And 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**

**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

## What is Database Per Microservice Pattern? How does it work?

The Database per Microservice pattern is a Microservices architecture pattern that involves creating a separate database for each microservice. In this pattern, each microservice has its own database, which allows for greater *scalability, flexibility, and autonomy*.

With this pattern, each Microservice has full control over its own data, which allows it to make independent decisions about how to store, retrieve, and manage its data.

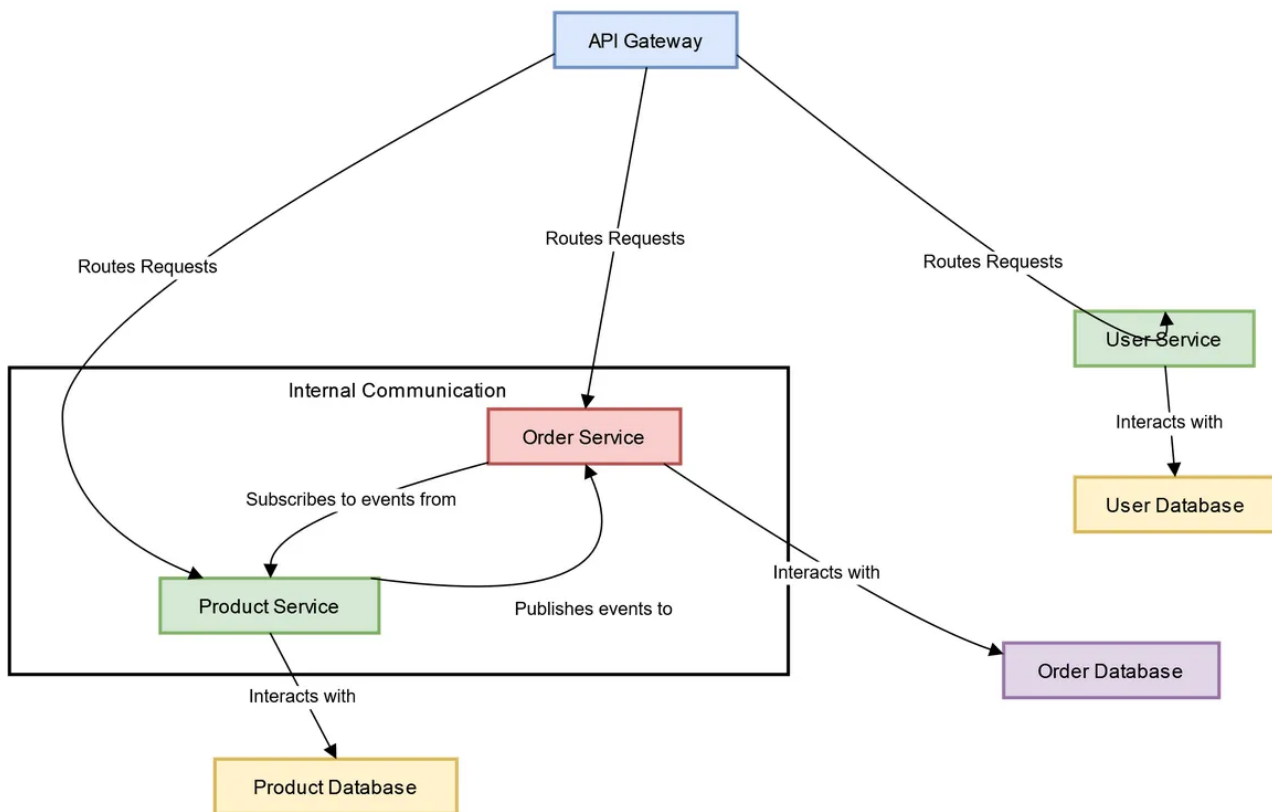
Each microservice **can choose the best database technology for its specific needs**, rather than being forced to use a single database technology for the entire system.

The pattern also **helps to reduce the risk of failure** in the system by removing single point dependency. Since each microservice has its own database, a failure in one microservice will not necessarily affect the rest of the system.

This also allows for easier maintenance, upgrades, and changes, as well as greater fault tolerance and resilience, one of the 10 things developer should keep in mind while designing microservices.

Here is how this pattern looks in practice, you can see that each service has its own database, `OrderService` is interacting with order database, User service has user database and Product Service also has product database.

## Database Per Microservices Pattern



Overall, the Database per Microservice pattern offers many benefits for microservices architecture, including scalability, flexibility, autonomy, and resilience. However, it also requires careful planning and coordination to ensure that data is properly managed and shared between microservices when necessary.

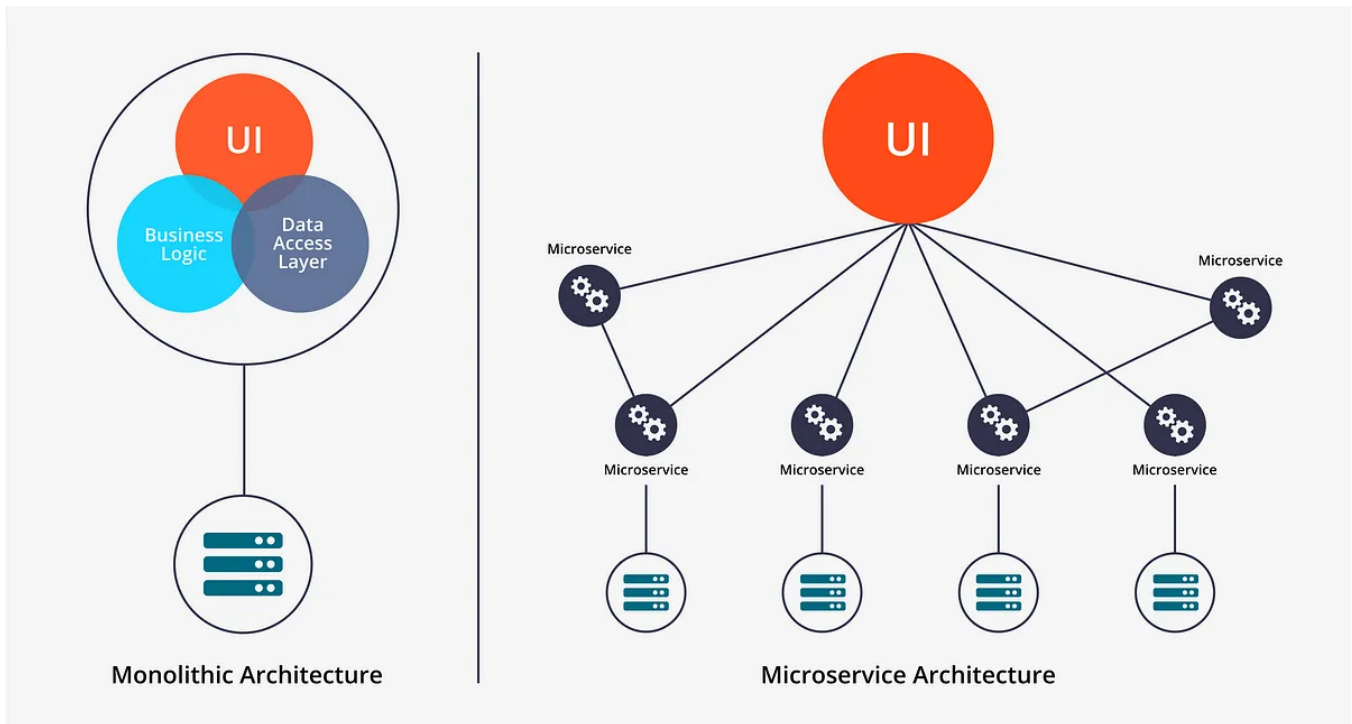
Now that we know what is Database Per Microservice Pattern, let's deep dive into it to understand it better.

### What Problem does Database per Microservice Pattern Solve?

The Database per Microservice design pattern solves the problem of managing data storage for microservices in a distributed system. In a microservices architecture, each microservice is responsible for a specific business capability and has its own data storage requirements.

Traditional monolithic applications usually have a single database that manages all data, but in a microservices architecture, managing data for multiple microservices in a single database can become complex and can cause issues such as tight coupling between microservices, difficulty in scaling and maintaining the database, and potential data integrity issues.

The Database per Microservice design pattern addresses these issues by allowing each microservice to have its own dedicated database, which provides better scalability, maintainability, and flexibility.



### When to use Database Per Microservice Pattern? A real world example

Suppose there is an e-commerce platform like Amazon.com with multiple microservices that handle various functionalities such as user authentication, order management, and inventory management. Each of these microservices requires its own database to store and manage the relevant data.

With the Database per Microservice Pattern, the e-commerce platform can ensure that each microservice has its own database, which helps to isolate data concerns, reduces coupling between services, and provides greater flexibility in scaling and deployment.

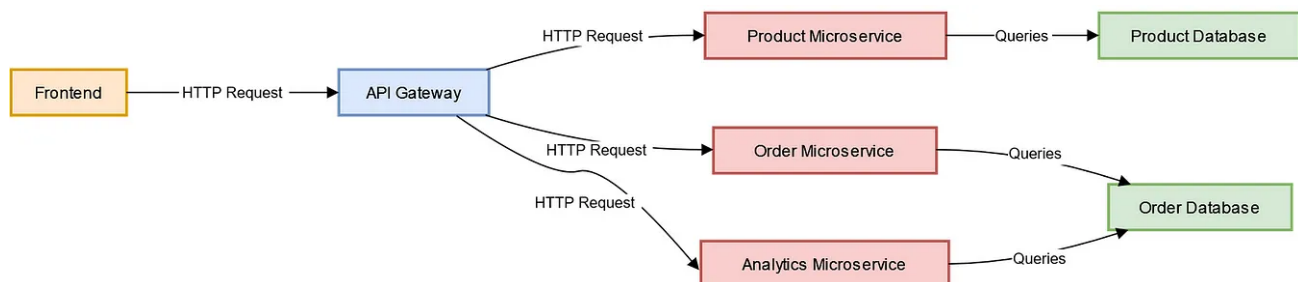
For instance, if the order management microservice experiences a surge in traffic, it can be scaled independently of the other services, since it has its own database.

Additionally, with the Database per Microservice Pattern, each microservice can choose the best-suited database technology for its specific needs.

For example, the **inventory management microservice** might use a **NoSQL database** for its high scalability and performance, while the **user authentication microservice** might use a **relational database** for its data consistency and transaction support.

Overall, the Database per Microservice Pattern can be a valuable approach to microservice architecture, particularly for complex applications that require multiple services to function.

Here is a diagram which shows how this pattern look like in a Microservice architecture:



In this scenario, an **API Gateway** sits at the front of the system and routes HTTP requests from the front-end to the appropriate microservices. The Product and Order microservices handle requests related to products and orders, respectively.

Each microservice has its own database (**Product Database** and **Order Database**), and the microservices make queries to their respective databases. An Analytics microservice also exists, which queries the Order Database to generate reports, which may seem counter intuitive but if Analytics Microservices is just reading from database then its still fine.

### What are Pros and Cons of Database Per Microservice Pattern?

Here are some of the pros and cons of using the Database per Microservice pattern:

#### Pros:

##### 1. Improved scalability

By having a separate database for each microservice, it becomes easier to scale individual services as per demand, without affecting others.

## 2. Increased autonomy

Each microservice is responsible for its own data, which means teams can work autonomously and independently.

## 3. Better performance

Since each microservice has its own database, it can use a database technology optimized for its specific needs, resulting in better performance.

## 4. Easier to maintain

Since each microservice has its own database, changes to one microservice won't affect others, making it easier to maintain and modify the system.

### Cons:

#### 1. Increased complexity

Managing multiple databases can be complex, and require a lot of effort to set up and maintain.

#### 2. Data consistency issues

Having multiple databases means that ensuring data consistency across all microservices can be challenging, and requires additional effort.

#### 3. Higher cost

Having multiple databases can lead to higher costs, both in terms of hardware and software licenses.

#### 4. Potential data duplication

Storing data in multiple databases can lead to data duplication, which can lead to inconsistencies and confusion.

It's important to note that the **suitability of this pattern depends on the specific requirements of the system being designed**, and should be evaluated carefully before implementation.

## 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

The **Database per Microservice pattern** offers several benefits, including increased scalability, flexibility, and fault tolerance. However, it also comes with some drawbacks, such as increased complexity and potential data inconsistency across microservices. Therefore, it is essential to carefully evaluate the needs of your system and consider the trade-offs before adopting this pattern.

The **database per microservice pattern** is particularly suitable for large and complex systems with a high degree of data isolation requirements. It is also an excellent choice for organizations that need to scale their applications quickly and frequently. However, smaller and less complex systems may not benefit from this pattern and may find it overly complex and costly to implement.

In summary, the **Database per Microservice pattern** is a powerful tool in the **microservices architecture toolbox**, but it should be used thoughtfully and with careful consideration of the system's requirements and constraints.

And , if you like this article and 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](https://medium.com)

Also, other **Microservices** articles you may like to explore:

### **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

### **What is difference between @Component and @Bean annotation in Spring?**

Hello folks, if you are preparing for question then you must prepare for questions like difference between X and Y like...

medium.com

### **Difference between @Controller, @Service, and @Repository Annotations in Spring Framework?**

While all three are stereotype annotation in Spring and can be use to represent bean where exactly they are used is the...

medium.com

Microservices

Programming

Java

Microservice Architecture

Software Engineering



Follow



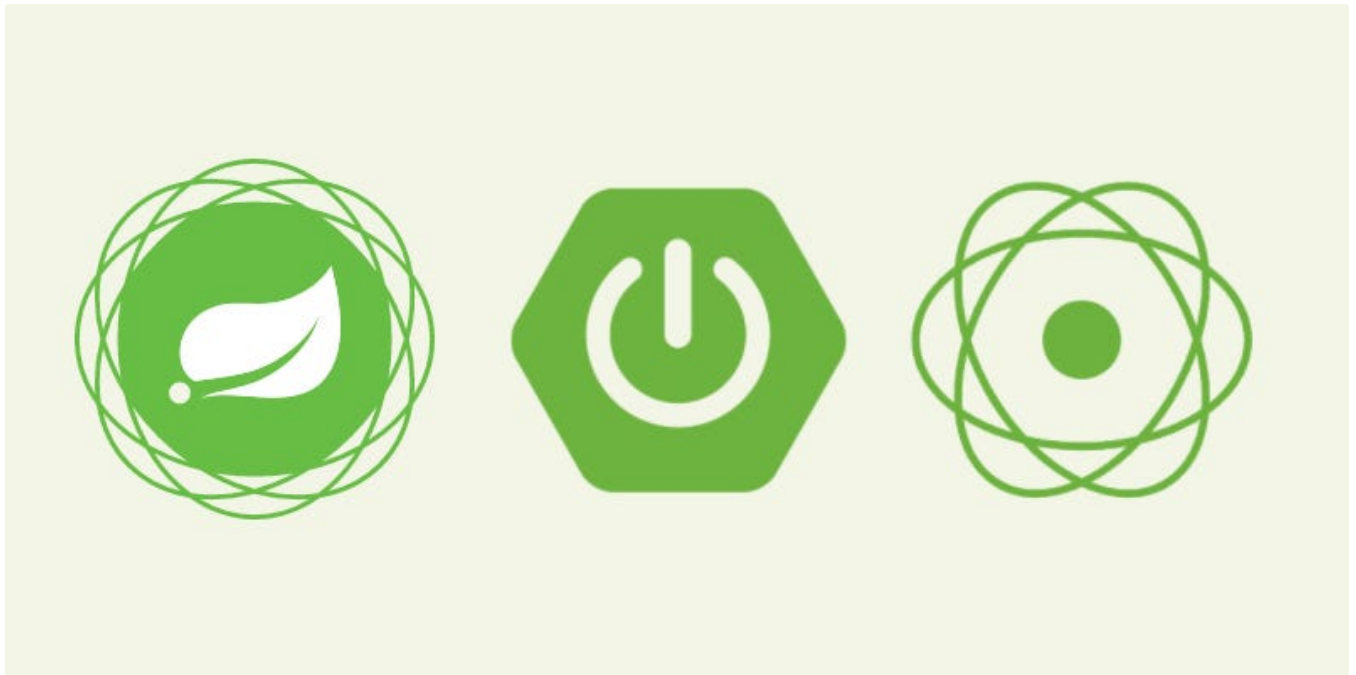
## Written by Soma

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](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





 Ajay Rathod in Javarevisited

## Comprehensive Spring-Boot Interview Questions and Answers for Senior Java Developers: Series-25

Greetings everyone,

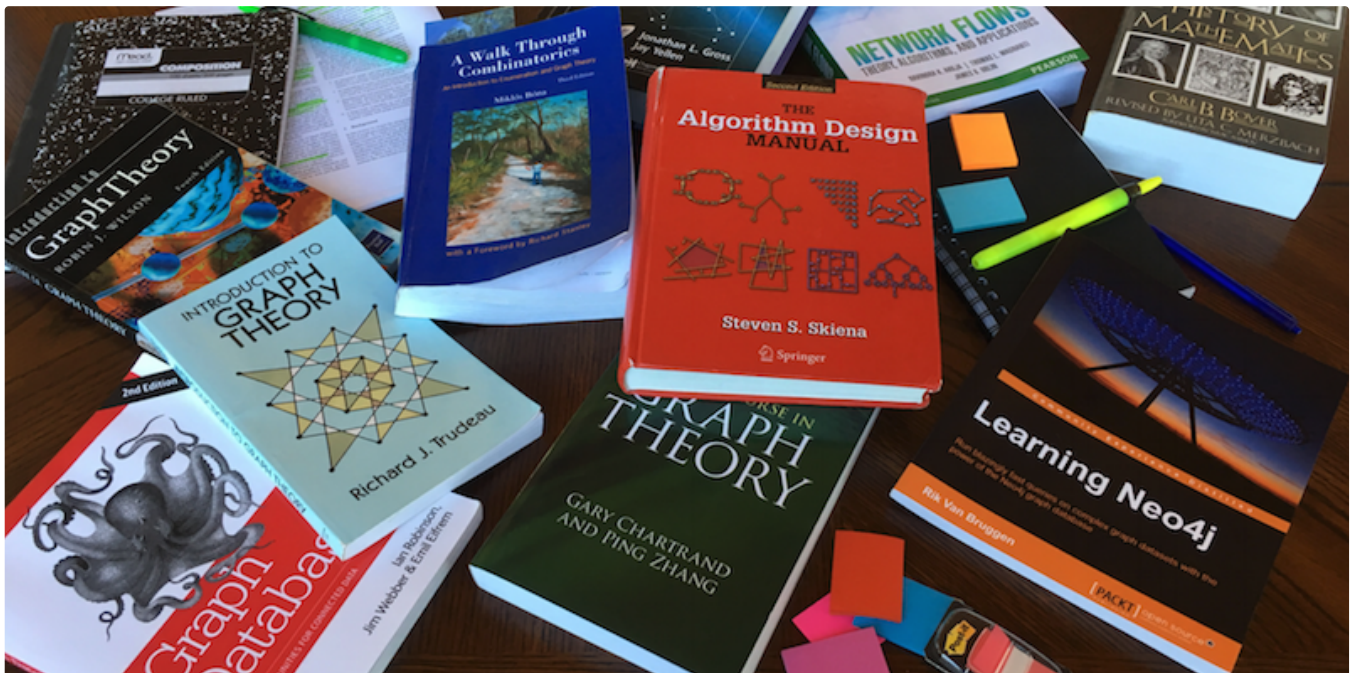
9 min read · 6 days ago




75



1



 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

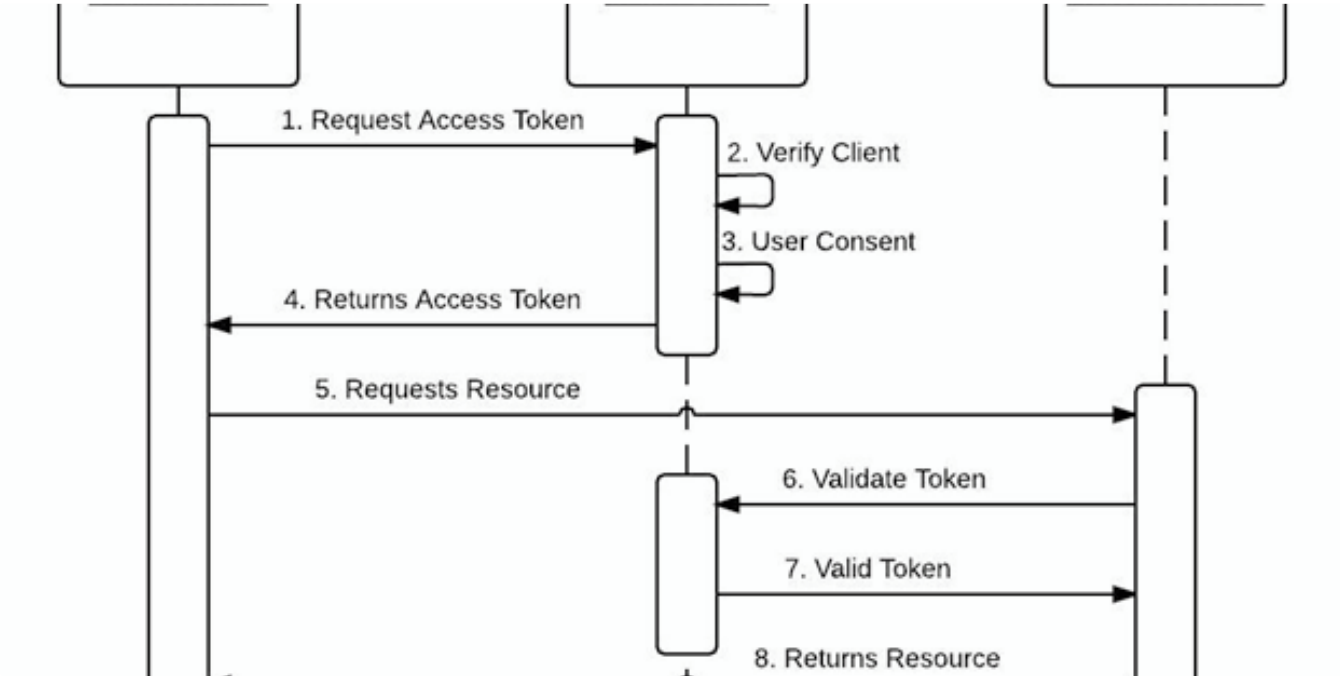
 4






- See all from Soma
- See all from Javarevisited

Recommended from Medium




 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

 31

 1







Anto Semeraro in Level Up Coding

# Microservices Orchestration Best Practices and Tools

Optimizing microservices communication and coordination through orchestration pattern with an example in C#

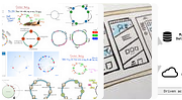
🌟 · 10 min read · Mar 27



69



## Lists



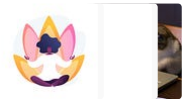
### General Coding Knowledge

20 stories · 201 saves



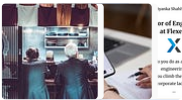
### It's never too late or early to start something

13 stories · 69 saves



### Stories to Help You Grow as a Software Developer

19 stories · 269 saves



### Leadership

35 stories · 91 saves

 Md Rasheduzzaman

## Microservices Event-Driven Architecture with RabbitMQ and Docker Containers

In the fast-paced and ever-evolving world of software development, the need for scalable, resilient, and efficient architectures is...

4 min read · 5 days ago



12



Brian NQC



## Backend 101 —A Guide to OpenAPI and API-First Approach

Learning outcomes: By the end of this blog, you should be able to:

11 min read · Feb 20



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



 Rapidcode Technologies

# The Twelve-Factor App: Best Practices for Cloud-Native Applications

Introduction

13 min read · Aug 5

 1 

See more recommendations