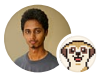
Service Discovery Pattern in Microservices

How to Use the Service Discovery Pattern in Microservices



Chameera Dulanga · Follow
Published in Bits and Pieces
7 min read · Jun 15, 2022

Listen Share

Microservice architecture is widely used in modern applications, and it has more advantages than those found in monolithic architecture. However, Microservices also come with several challenges. One such challenge is finding the locations of services since they often change dynamically.

So, in this article, I will discuss how we can overcome this by using the service discovery pattern.

Why We Need Service Discovery Pattern?

In traditional monolithic applications, services run in specific, fixed locations. So, it is easy to keep track of their locations, and clients can easily find and invoke services.

However, when it comes to microservices, their network locations are dynamically assigned and often change due to features like auto-scaling. This behavior makes it hard to keep track of microservice locations beforehand.

As a solution, the service discovery pattern was introduced, and it allows developers to find the network locations of microservices without injecting or coupling services.

What is Service Discovery Pattern?

The service discovery pattern is mainly used in microservices applications to find the network locations of microservices.

The service discovery pattern uses a centralized server named "**service registry**" to maintain a global view of microservices' network locations. Microservices update their locations in the service registry at fixed intervals. Clients can connect to the service registry and fetch the locations of microservices.

There are 2 main service discovery patterns available to implement service discovery for microservices.

- **Client-side service discovery**
- **Server-side service discovery**

But, before discussing them individually, let's see how service discovery works to get a better understanding.

How Does Service Discovery Work?

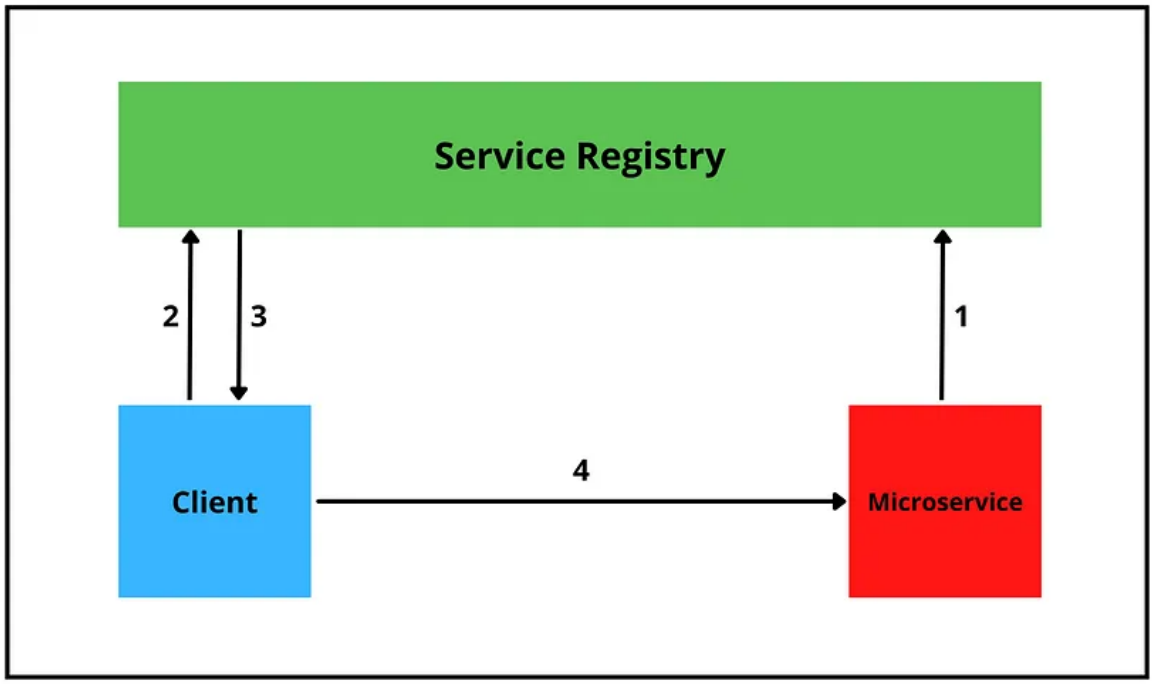
As mentioned, there are 3 participants in the service discovery pattern:

- **Service registry**
- **Client**

- **Service consumers (Microservices)**

The service registry keeps records of the network locations of the microservices. Microservice instances register themselves and provide their locations to the service registry. Then, clients can find the locations of microservices using the service registry and directly call them to fulfill its need.

To get a better understanding, let's consider a simple microservice architecture with 2 services:



Here, we can break the service discovery communication flow into 4 parts:

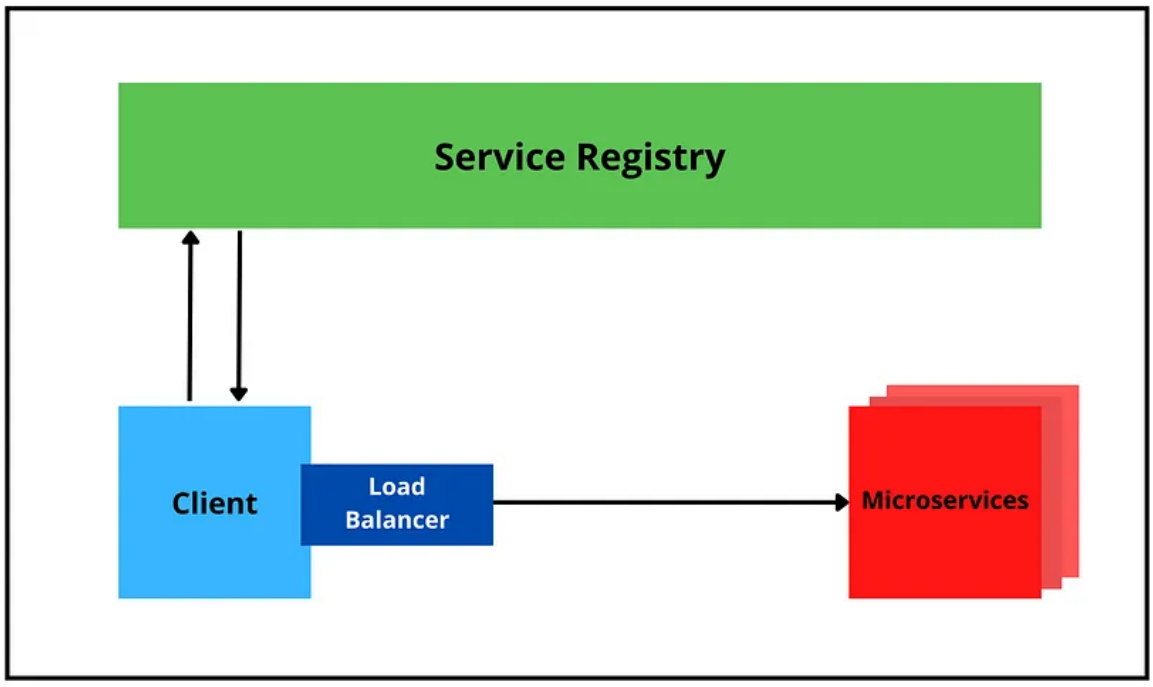
1. First, the service provider registers its location in the service registry.
2. The client looks for relevant service locations in the service registry.
3. The service registry returns the location of the required microservice.
4. The client directly calls the microservice.

As you can see, the service discovery pattern is pretty straightforward. However, there can be slight changes in the above-described process in its implementation.

The Client-side Discovery Pattern

In the Client-side service discovery pattern, the client is responsible for finding the network locations of microservices and loading balancing requests between them.

First, the client queries the service registry and retrieves the locations. Then, the client uses its dedicated load balancer to select a microservice and sends the request.



Advantages of Client-side Service Discovery

- It is straightforward and easy to understand.
- The service registry is the only moving part.
- The client knows the locations of microservices before sending the request.
- The client can make intelligent load balancing decisions with its dedicated load balancer.

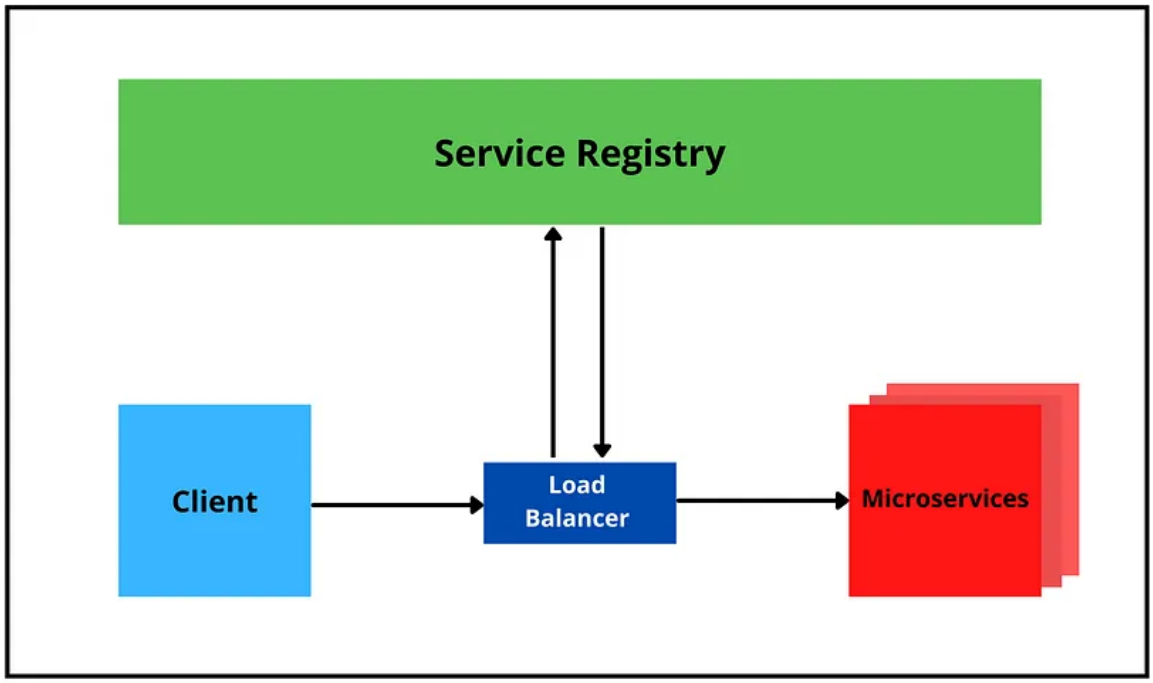
Disadvantages of Client-side Service Discovery

- The client is responsible for implementing service discovery logic.
- The client is coupled with the service registry.
- Need to implement service discovery logic for each language used by your clients.

The Server-side Discovery Pattern

The Server-side discovery pattern solves one of the significant issues in Client-side discovery by decoupling the client and the services registry. In this approach, the client does not have a dedicated load balancer. Instead, the load balancer acts as a middle man and is responsible for communicating with the service registry.

First, the client requests a microservice through the load balancer. Then, the load balancer queries the service registry and finds the location of the relevant microservice. Finally, the load balancer routes the request to the microservice.



This pattern is widely used in modern applications since both clients and microservices are independent of the service registry. Most importantly, you don't need to implement the Server-side discover load balancers from scratch since many deployment environments provide load balancers.

For example, you can use AWS Elastic Load Balancer or proxies on the Kubernetes environments as Server-side discovery load balancers.

Advantages of Server-side Service Discovery

- The service registry is decoupled from the client.
- There is no need to implement service discovery logic for each language your clients use.

- Can use load balancers provided by deployment environments.

Disadvantages of Server-side Service Discovery

- If the deployment environment does not provide, developers need to create and manage load balancers.

What is Service Registry?

Throughout the article, I have mentioned that the service registry is responsible for keeping the locations of each microservice. So, it is essential to know how the service registry works to understand the service discovery pattern completely.

The service registry is a database containing all available microservices' network locations. Therefore, it should be highly available and continuously updated. So, it is essential to have a reliable mechanism to continuously update and maintain data consistency in a service registry.

In general, a microservice is registered in the service registry when the service starts up. Then, it continuously updates its registration using a heartbeat mechanism. Finally, when the microservice instance terminates, the registration is removed from the service registry.

There are 2 approaches to handling the registration process in the service registry:

- Self-registration.
- Third-party registration.

Both these approaches have several advantages and disadvantages. So, let's discuss these 2 approaches in detail to understand them better.

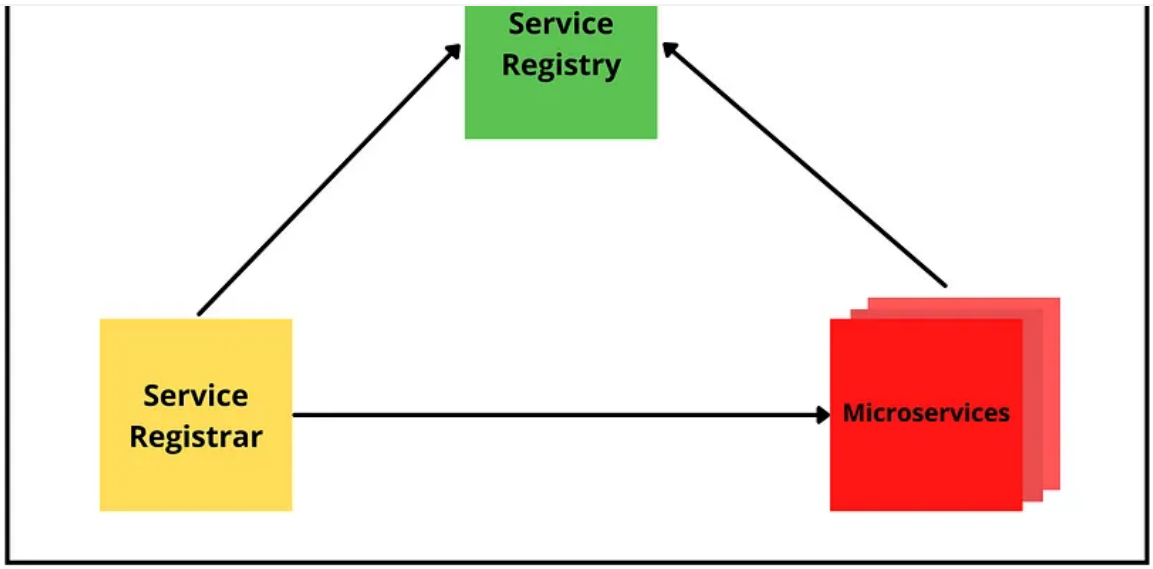
The Self-Registration

In self-registration, the microservices are responsible for registering themselves in the service registry. Once the microservice is registered, it will keep sending heartbeats to ensure that the registration does not expire.

This approach is straightforward and does not require any third parties to be involved. On the other hand, service registry and microservices are coupled together. Therefore, it can lead you to implement the registration logic in multiple languages to be compatible with all the microservices.

The Third-Party Registration

In this approach, a new component named service registrar is responsible for registering and de-registering microservices. In addition, the service registrar tracks the location changes of microservices by polling the deployment environment or subscribing to the events.



Many modern deployment environments provide a built-in component as the service registrar. For example, if you create an EC2 instance using an auto-scaling group, it will automatically

register with an Elastic Load Balancer.

Since microservices are decoupled from the service registry, developers don't need to spend much time implementing the service registration logic. However, if the deployment environment does not provide the service register, developers need to implement it and make sure it is highly available.

Final Thoughts

I have discussed the service discovery pattern in microservices, how it works, and its different components.

The service discovery pattern is beneficial in modern microservices architecture to find microservices' network locations. In addition, many modern deployment platforms support this pattern by default to make developers' work easier.

I hope you have found this article very helpful. Thank you for Reading!

Bit: Feel the power of component-driven dev

Say hey to **Bit**. It's the #1 tool for component-driven app development.

With Bit, you can create any part of your app as a "component" that's composable and reusable. You and your team can share a toolbox of components to build more apps faster and consistently together.

- Create and compose "**app building blocks**": UI elements, full features, pages, applications, serverless, or micro-services. With any JS stack.
- Easily **share, and reuse** components as a team.
- Quickly **update components** across projects.
- Make hard things simple: Monorepos, design systems & micro-frontends.

Try Bit open-source and free→

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

How to reuse React components across your projects

Finally, you completed the task of creating a fantastic input field for the newsletter in your app. You are happy with...

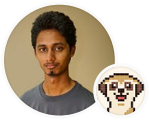
bit.cloud

Painless monorepo dependency management with Bit

Simplify dependency management in a monorepo to avoid issues with phantom dependencies and versions. Learn about...

bit.cloud

Microservices Design Patterns Web Development Software Development Programming



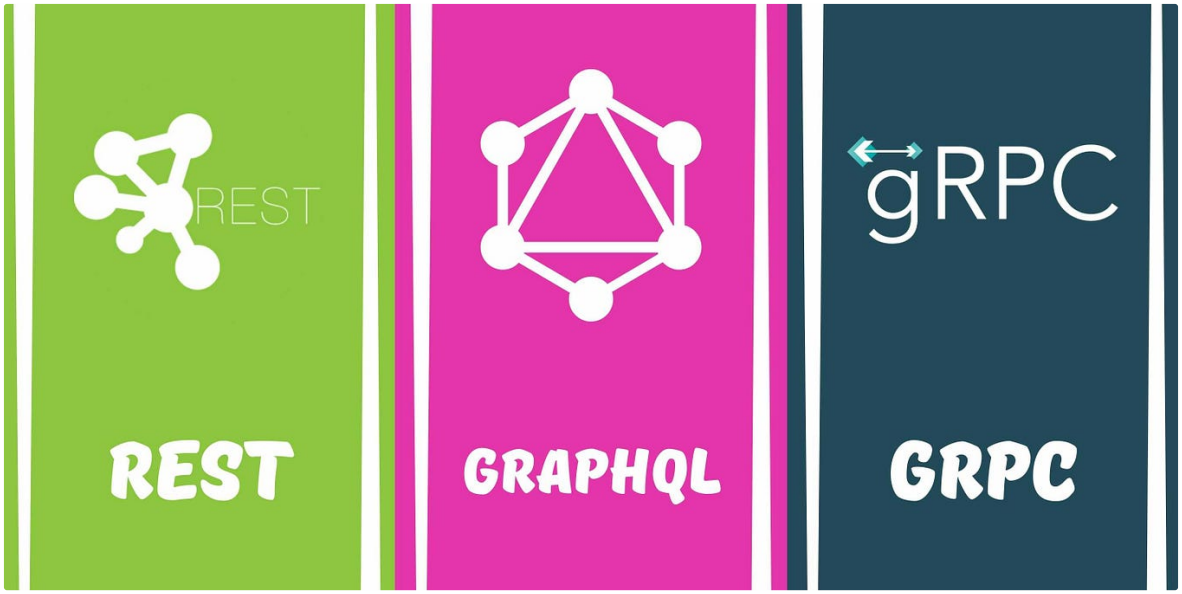
Follow

Written by Chameera Dulanga

3K 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

REST vs GraphQL vs gRPC

In-depth comparison of the 3 most popular API development technologies

6 min read · Jun 10, 2022

3.3K 34

Anto Semeraro in Bits and Pieces

JavaScript Optimization Techniques for Faster Website Load Times: An In-Depth Guide

Master JavaScript optimization to enhance website performance: minimize file sizes, reduce requests, leverage caching & async loading, and...

16 min read · Apr 14

524 3

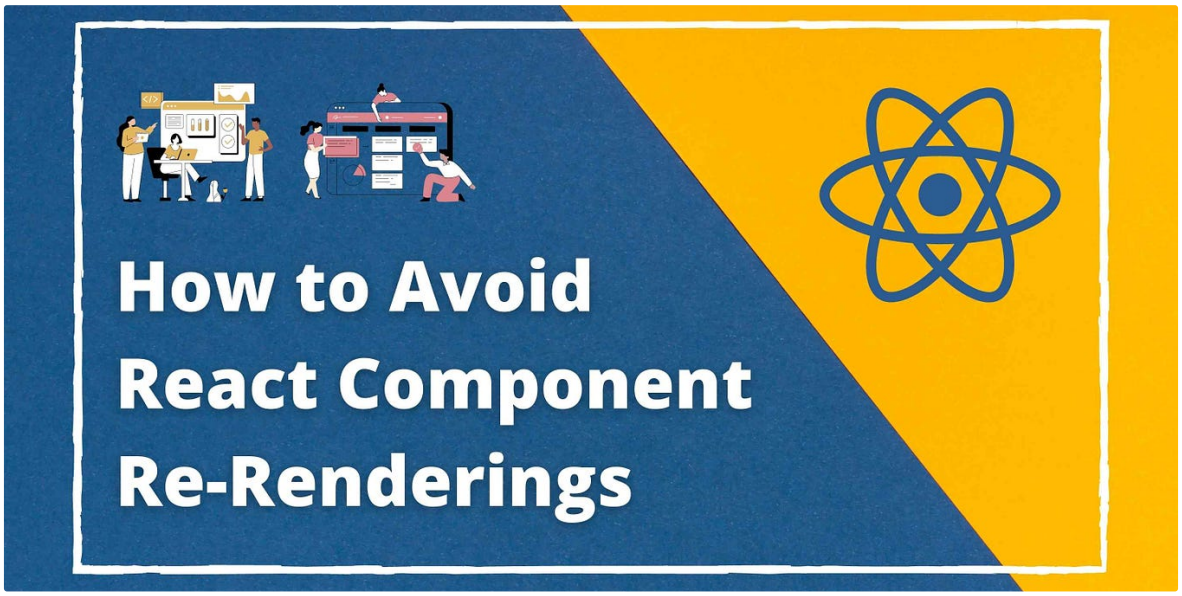
Fernando Doglio in Bits and Pieces

Node.js Just Released Version 20! WTH?!

Let's see what's new with Node v20!

8 min read · Apr 24

566 4



Chameera Dulanga in Bits and Pieces

5 Ways to Avoid React Component Re-Renderings

How to Avoid Unnecessary Re-rendering in React

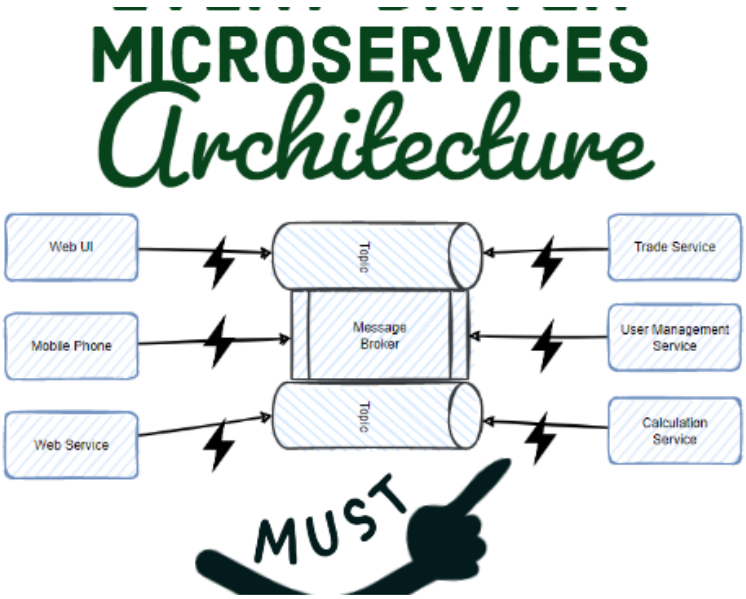
· 6 min read · Jan 12, 2022

1.6K 6

See all from Chameera Dulanga

See all from Bits and Pieces

Recommended from Medium



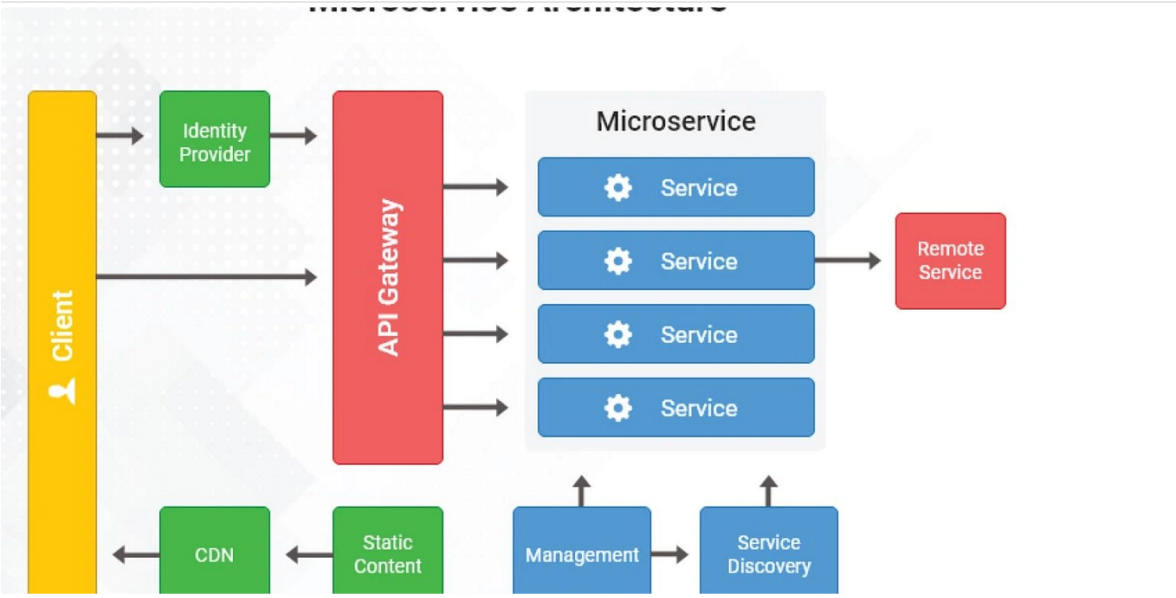
Farhad Malik in FinTechExplained

What Is Event-Driven Microservices Architecture?

Designing Modern Event-Driven Microservices Applications With Kafka And Docker Containers Suitable For All Levels

· 13 min read · Dec 6, 2022

123 2



Dineshchandgr - A Top writer in Technology in Javarevisited

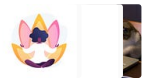
Do you know about Microservices and their Design Patterns?

Hello everyone. In this article, we are going to see about Microservice Architecture and how it is different from a Monolithic application...

· 8 min read · Feb 3

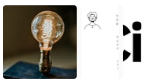
420 2

Lists



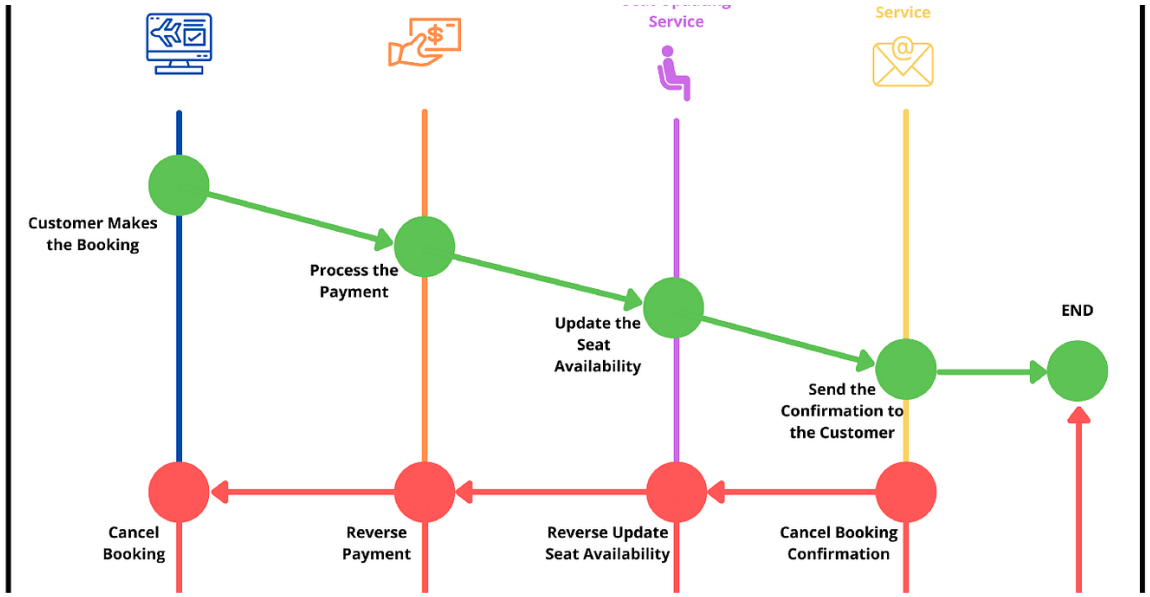
Stories to Help You Grow as a Software Developer
19 stories · 22 saves

Leadership
30 stories · 9 saves



Good Product Thinking
11 stories · 25 saves

Stories to Help You Level-Up at Work
19 stories · 19 saves



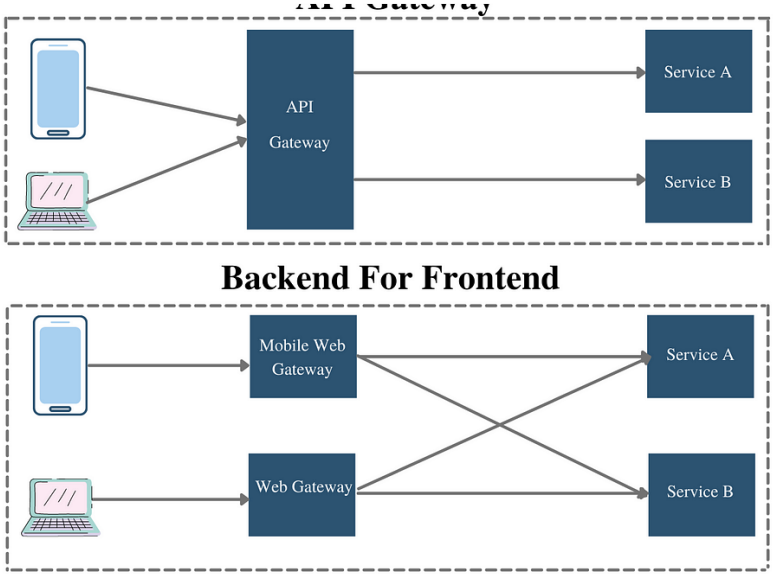
Soma in Javarevisited

What is SAGA Pattern in Microservice Architecture? Which Problem does it solve?

SAGA is an essential Micro service Pattern which solves the problem of maintaining data consistency in distributed system

✦ · 6 min read · Jan 31

315 1



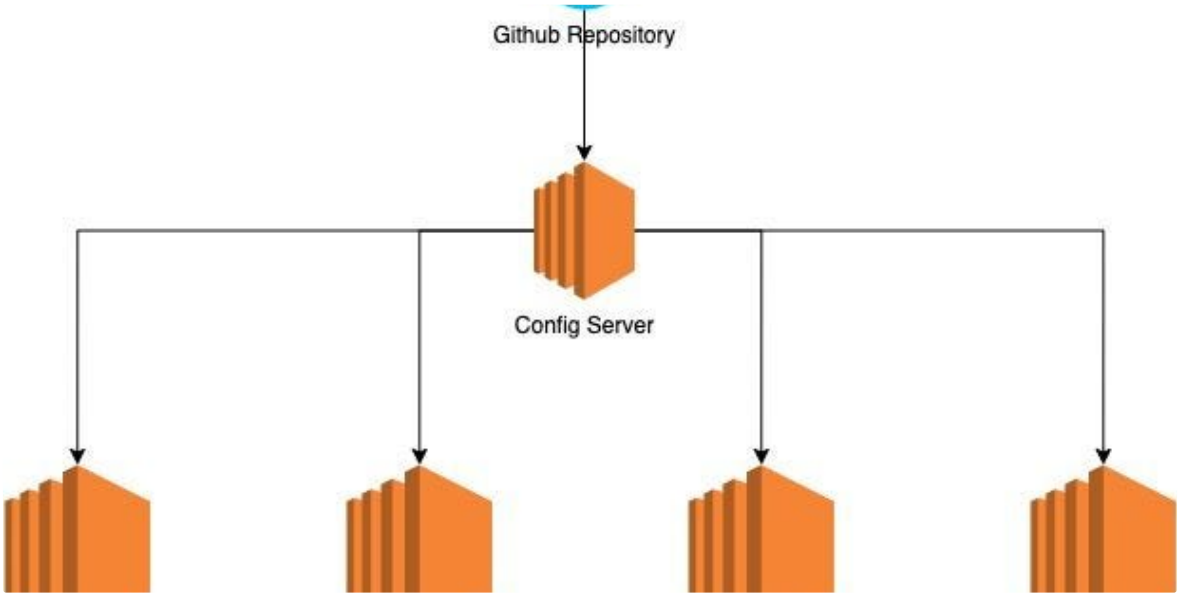
Dineshchandgr - A Top writer in Technology in Javarevisited

API Gateway vs Backend For Frontend (BFF)—Everything you need to know

Hello everyone. In this article, we are going to see what is an API Gateway and how it is used for Client-Server communication in a...

✦ · 6 min read · Dec 2, 2022

136 5



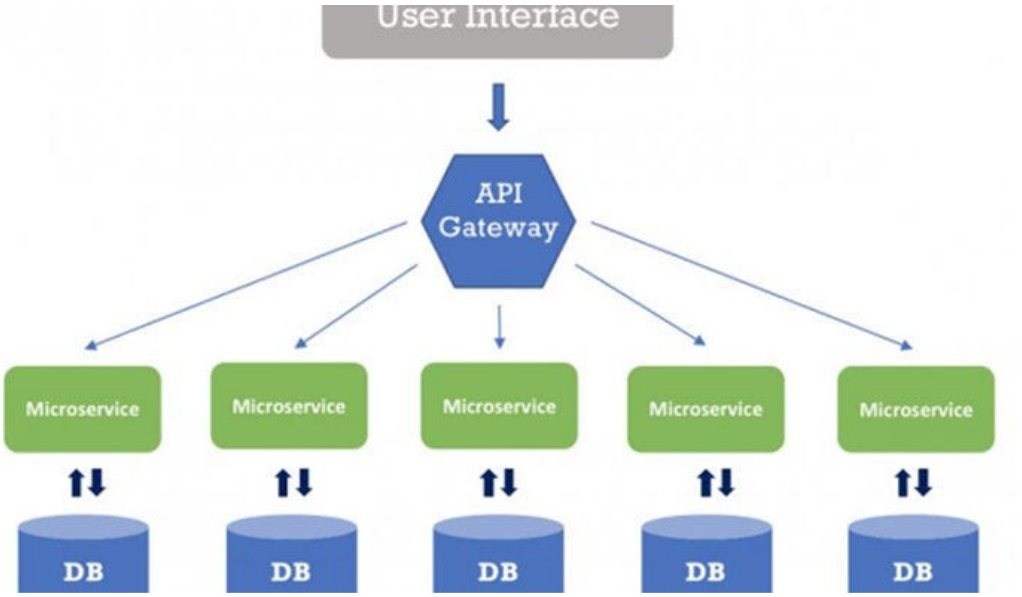
Dineshchandgr - A Top writer in Technology in Javarevisited

Why do we need to externalize the configuration in Microservices Architecture?

Hello everyone. In this article, we are going to see what is a configuration and look at the problems in maintaining configuration for a...

✦ · 7 min read · Dec 28, 2022

154 4



Soma in Javarevisited

50 Microservices Design and Architecture Interview Questions for Experienced Java Programmers

Preparing for Senior Java developer role where Microservices skill is required? Here are 50 questions which you should know before going...

· 11 min read · Jan 14

344 5