Open in app ↗

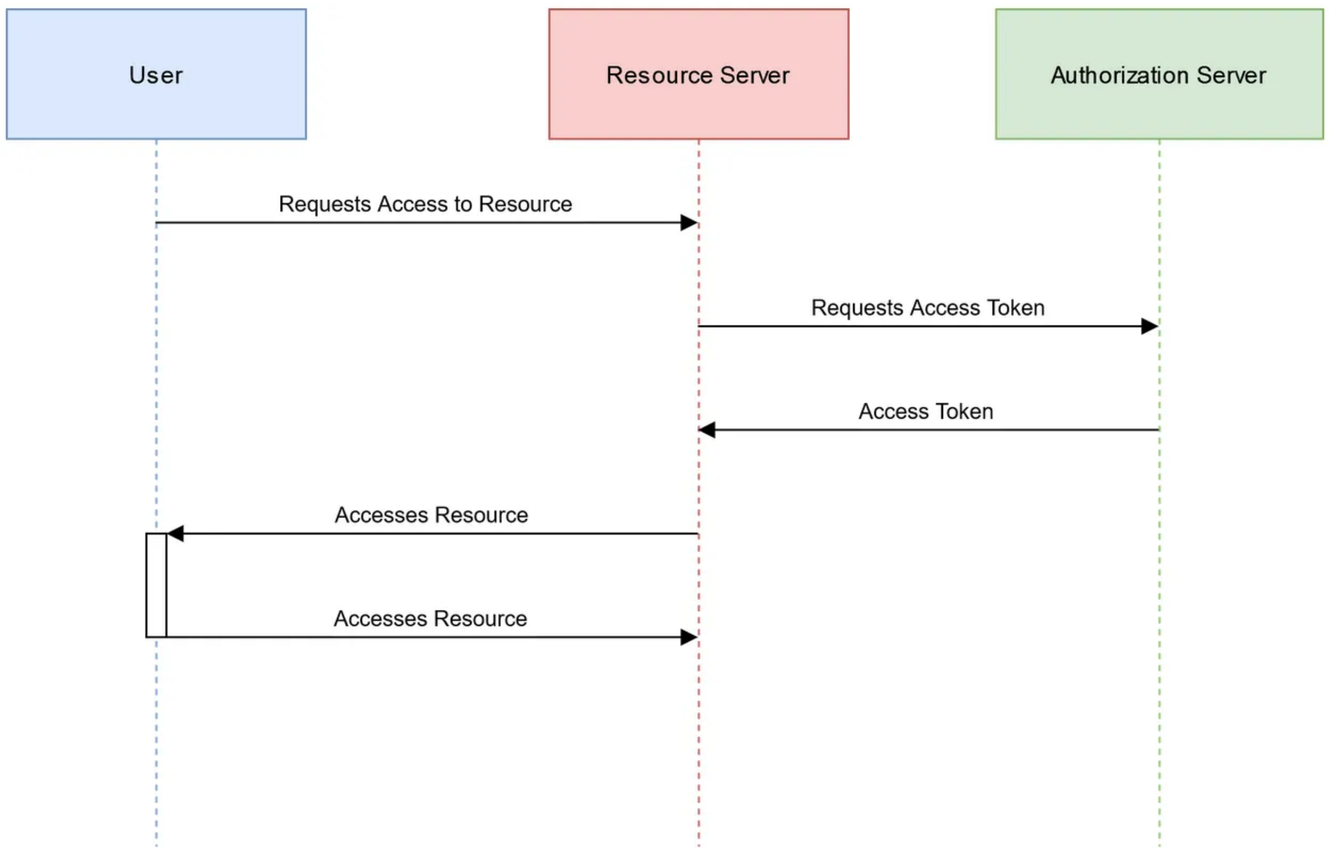
Member-only story

Securing Microservices with OAuth2 and Spring Security

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

Soma · Follow
Published in Javarevisited
10 min read · Apr 7

Listen Share More



Hello folks, as **Microservices architecture** become more and more popular in modern application development particularly in cloud, ensuring robust security becomes a critical concern. It's also one of the *popular Microservice interview question* which is often asked to experienced programmers and developers.

One common approach to securing microservices is by leveraging the **OAuth2 protocol** in conjunction with **Spring Security,** a powerful and widely used security framework for Java applications.

Security is also one of the main concern while designing Microservices as mentioned in my earlier article, 10 things to keep in mind while designing Microservices.

In the last few articles you have learned popular Microservice design patterns like **Event Sourcing**, **SAGA**, Database Per Microservices, **API Gateway**, **Circuit-Breaker**, and **CQRS** and also learn *about the best practices to design Microservices* and in this article, we will explore the concept of securing microservices using OAuth2 and Spring Security.

We will also delve into the fundamentals of OAuth2, understand how it works with Spring Security, and explore best practices for securing microservices in a distributed environment.

So, let's dive in and learn how to effectively secure microservices with OAuth2 and Spring Security!

1. Why Security is Important in Microservices?

In Microservices architecture, each service is responsible for its own data storage and processing. As a result, securing the communication between these services is critical to maintaining data confidentiality, integrity, and availability.

Furthermore, since Microservices are distributed and run in different environments, the attack surface increases, making them more vulnerable to security threats. Therefore, implementing security measures to protect the Microservices is essential.

By implementing OAuth2 and Spring Security for your Microservices, you can not only ensure proper authentication and authorization, but also protect sensitive data, and build secure and scalable microservices-based applications.

OAuth2 is an open-standard authorization protocol that allows secure access to resources without sharing login credentials. It is widely used to authenticate and authorize user access to APIs.

Spring Security is a powerful and highly customizable security framework for Java applications. It provides authentication and authorization services and integrates well with other Spring modules.

Together**, OAuth2 and Spring Security can be used to secure Microservices communication and protect them from unauthorized access.**

In the following sections, we will dive deeper into OAuth2 and Spring Security and how they can be used to secure Microservices in Java applications.

2. Understanding OAuth2

Now, let's deep dive into OAuth2 protocol and understand how it can secure your Microservices and why its popular in today's digital world.

2.1. Explanation of OAuth2 protocol

OAuth2 is a protocol that enables secure authorization and authentication for applications. It provides a way for users to grant third-party applications access to their resources without sharing their login credentials.

The OAuth2 protocol involves four roles:

1. the resource owner (user),
2. the client (application),
3. the resource server (API),
4. and the authorization server (identity provider).

2.2. OAuth2 grant types

OAuth2 supports several grant types for different use cases. The most common grant types are:

- **Authorization Code**: Used for web applications that run on a server and need to access resources on behalf of a user.
- **Implicit**: Used for single-page applications that run in a web browser and need access to resources without the need for a server.
- **Resource Owner Password Credentials:** Used for trusted applications that require direct access to a user's resources.
- **Client Credentials**: Used for applications that need to access their own resources.

2.3. OAuth2 components

OAuth2 involves several components that work together to enable secure access to resources. The main components are:

- **Authorization Server:** Handles user authentication and authorization and provides access tokens to clients.
- **Resource Server:** Hosts the protected resources that clients want to access.
- **Client:** Requests access to resources on behalf of the user.
- **Resource Owner:** Owns the protected resources and grants permission to clients to access them.

2.4. OAuth2 authentication flow

The OAuth2 authentication flow involves several steps:

1. The client requests authorization from the user.
2. The user authenticates with the authorization server and grants permission to the client.
3. The authorization server issues an access token to the client.
4. The client uses the access token to request resources from the resource server.
5. The resource server verifies the access token and responds with the requested resources.

In the next section, we will discuss how Spring Security can be used to implement OAuth2 in Java applications.

3. Understanding Spring Security

Spring Security is a popular security framework for Java applications, including microservices. It provides a robust and flexible security layer that can be easily integrated into your microservices architecture.

At its core, **Spring Security is an authentication and authorization framework**. It helps to secure your application by controlling access to resources based on user roles and permissions.

Spring Security provides a range of components, including authentication and authorization filters, user management, and access control. These components can be used to build a comprehensive security system for your microservices.

One of the key features of Spring Security is its **modular architecture,** which allows you to pick and choose the components that are most relevant to your application's security needs. This makes it easy to integrate Spring Security into your existing microservices architecture.

Spring Security also provides support for various authentication mechanisms, such as form-based authentication, OAuth2, and OpenID Connect. This means that you can choose the authentication mechanism that's best suited to your application's needs.

In summary, Spring Security is a powerful security framework that provides a range of components and features to help you secure your microservices. Its modular architecture and support for various authentication mechanisms make it a popular choice for Java developers.

4. How to Secure Microservices with OAuth2 and Spring Security?

Now that you have solid understanding of OAuth2 and Spring Security, let's explore how to secure microservices using these technologies.

Microservices architecture is a distributed system, which means that it's more vulnerable to security threats than a monolithic system. With microservices, the attack surface increases, making it necessary to implement a robust security system.

OAuth2 and Spring Security can be used together to secure microservices. OAuth2 provides a framework for authentication and authorization, while Spring Security provides the components necessary to implement this framework in your microservices.

To secure microservices using OAuth2 and Spring Security, you need to **implement the OAuth2 authorization server and resource server.** The authorization server issues access tokens to clients, while the resource server validates these access tokens and provides access to protected resources.

Spring Security provides the necessary components to implement the resource server. This includes the *OAuth2 authentication filter,* which validates access tokens, and the access decision manager, which controls access to protected resources.

One of the advantages of using OAuth2 and Spring Security to secure microservices is that they provide *a standardized way of implementing security*. This means that your microservices can be easily integrated with other systems that also use OAuth2 and Spring Security.

Steps to Implement OAuth2 and Spring Security in Microservices

Securing microservices is essential to protect the resources and data they provide. OAuth2 and Spring Security are two widely used tools for securing microservices. In this section, we will discuss how to implement OAuth2 and Spring Security in microservices.

4.1 Configure OAuth2 Authorization Server

Configuring OAuth2 Authorization Server The first step in implementing OAuth2 in microservices is to configure an authorization server.

The authorization server is responsible for issuing access tokens to clients after they are authenticated. In Spring Security, the `@EnableAuthorizationServer` annotation is used to configure the authorization server.

4.2 Configuring OAuth2 Resource Server

The next step is to configure the resource server to validate the access token provided by the client. The resource server is responsible for granting or denying access to the protected resources. In Spring Security, the `@EnableResourceServer` annotation is used to configure the resource server.

4.3 Adding OAuth2 Client Support to Microservices

The microservices must be able to communicate with the authorization server to obtain an access token. In Spring Security, the `OAuth2RestTemplate` class is used to make requests to the authorization server to obtain an access token.

4.4 Securing Microservices Endpoints with Spring Security

Once the microservices have obtained an access token, they must validate the token and authorize the client. In Spring Security, the `@PreAuthorize` annotation is used to secure the endpoints. The annotation specifies the roles or authorities required to access the endpoint.

4.5 Testing Microservices with OAuth2 and Spring Security

Testing microservices with OAuth2 and Spring Security is another crucial step to ensure the security of the microservices. In Spring Security, the `@WithMockUser` annotation is used to simulate a user during testing. The annotation specifies the roles or authorities of the simulated user.

5. Security Microservice with OAuth2 and Spring Security- Example

Now that we have seen the steps to secure Microservices, its time to put them into action.

First, we need to configure the OAuth2 authorization server in our application. We can do this by creating a class that extends the `AuthorizationServerConfigurerAdapter` :

```
@Configuration
@EnableAuthorizationServer
public class AuthServerConfig extends AuthorizationServerConfigurerAdapter {

    @Autowired
    private AuthenticationManager authenticationManager;

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Except
        clients.inMemory()
            .withClient("client")
            .secret("{noop}secret")
            .authorizedGrantTypes("password")
            .scopes("read", "write")
            .accessTokenValiditySeconds(3600);
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) thr
        endpoints.authenticationManager(authenticationManager);
    }

}
```

In this example, we're using an **in-memory client store** and a single client with a username of "client" and password of "secret". We're also using the "password" grant type and allowing access to the "read" and "write" scopes. The access token is set to expire after 3600 seconds.

Next, we need to configure the resource server that will protect our **OrderService**:

```
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/orders/**").authenticated()
            .and().sessionManagement().sessionCreationPolicy(SessionCreationPol
            .and().csrf().disable();
    }

}
```

In this example, we're using an `HttpSecurity` configuration to secure our `/orders/**` endpoint. We're requiring authentication for this endpoint and disabling CSRF protection. We're also using stateless sessions to avoid any session-related security issues.

Finally, we need to configure our **OrderService to use OAuth2 authentication:**

```
@RestController
@RequestMapping("/orders")
public class OrderController {

    @Autowired
    private RestTemplate restTemplate;

    @Value("${oauth2.resource.url}")
    private String resourceUrl;

    @GetMapping("/{id}")
    public Order getOrder(@PathVariable Long id, OAuth2Authentication auth) {
        String url = resourceUrl + "/orders/" + id;
        HttpHeaders headers = new HttpHeaders();
        headers.setBearerAuth(auth.getName());
        HttpEntity<Void> entity = new HttpEntity<>(headers);
        ResponseEntity<Order> response = restTemplate.exchange(url, HttpMethod.
        return response.getBody();
    }

}
```

In this example, we're using a `RestTemplate` to make a request to our OrderService. We're passing the access token from the authenticated `OAuth2Authentication` object as a Bearer token in the request headers. This ensures that the OrderService can validate the access token and authorize the request.

With this code example, we have secured our OrderService with OAuth2 and Spring Security, and we can be confident that our service is protected from unauthorized access.

6. Best practices for Microservices security with OAuth2 and Spring Security

Security is a crucial aspect of Microservices architecture, and it is essential to follow the best practices to ensure that Microservices are secure. Here are some best practices to secure Microservices with OAuth2 and Spring Security:

6.1 Use HTTPS for secure communication

HTTPS is an essential security measure that ensures that all communication between the client and the server is encrypted. It is crucial to use HTTPS for all Microservices endpoints to prevent unauthorized access to sensitive information.

6.2 Implementing two-factor authentication

Two-factor authentication is a security measure that adds an extra layer of security to the authentication process. It requires users to provide two forms of authentication, such as a password and a one-time code, to access a Microservice. Implementing two-factor authentication can help prevent unauthorized access to sensitive resources.

6.3 Limiting access to sensitive resources

It is essential to limit access to sensitive resources in Microservices. Access control mechanisms such as role-based access control (RBAC) and attribute-based access control (ABAC) can be used to limit access to sensitive resources to only authorized users.

6.4 Protecting against common attacks such as CSRF and XSS

Cross-site request forgery (CSRF) and cross-site scripting (XSS) attacks are common attacks that can compromise the security of Microservices. It is essential to implement measures to protect against these attacks, such as using CSRF tokens and input validation.

By following these best practices, Microservices can be secured against common security threats, and users can be confident that their data is safe and secure.

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 secure your Microservice using Spring Security framework in Java world.** Securing Microservices is a critical aspect of modern software development, and implementing OAuth2 and Spring Security can provide a robust and effective solution.

By following best practices and staying up to date with the latest security trends, Java developers can ensure that their Microservices are secure and reliable for their users.

And , if you like this Microservice and Spring Security article and 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

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

What is API Gateway Pattern in Microservices Architecture? What Problem Does it Solve?

The API Gateway can help in managing authentication, request routing, load balancing, and caching in Microservices...

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

Difference between @Controller and @RestController in Spring Boot and Spring MVC?

@Controller is used to declare common web controllers which can return HTTP response but @RestController is used to...

medium.com

Microservices Programming Software Engineering Java Spring Boot

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

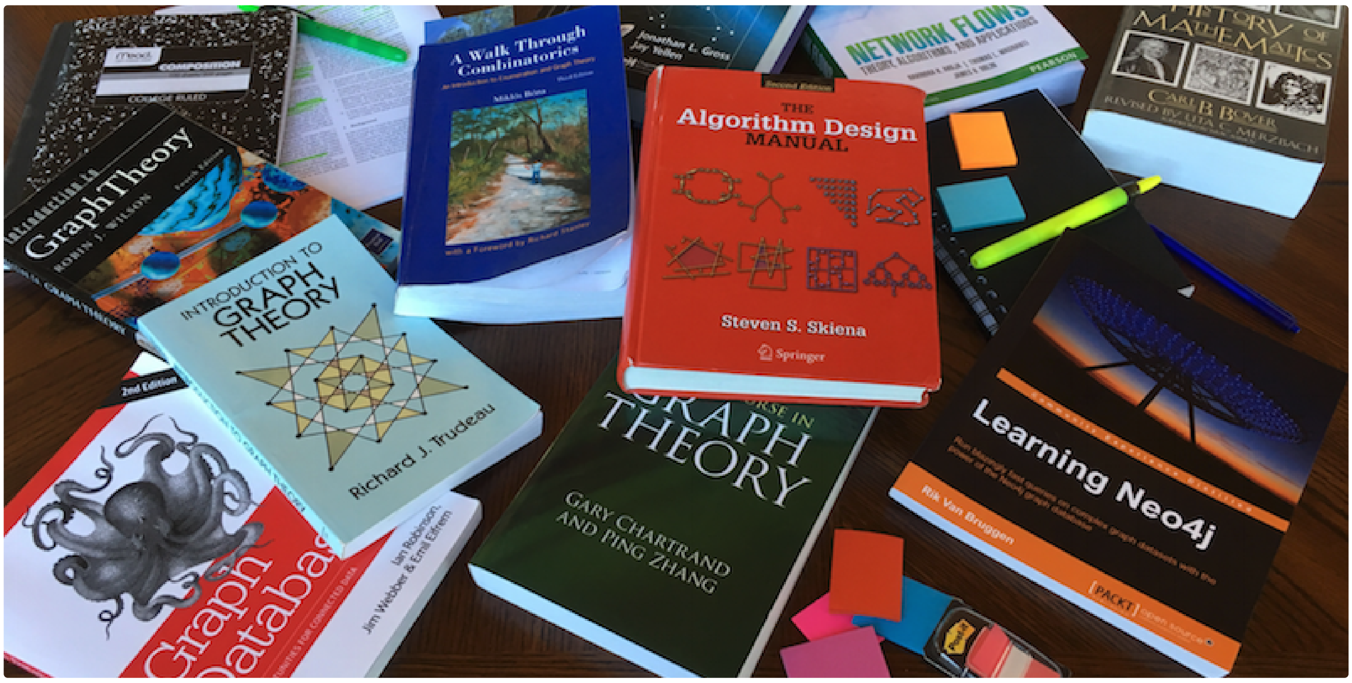
Ajay Rathod in Javarevisited

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

Greetings everyone,

9 min read · 6 days ago

75 1



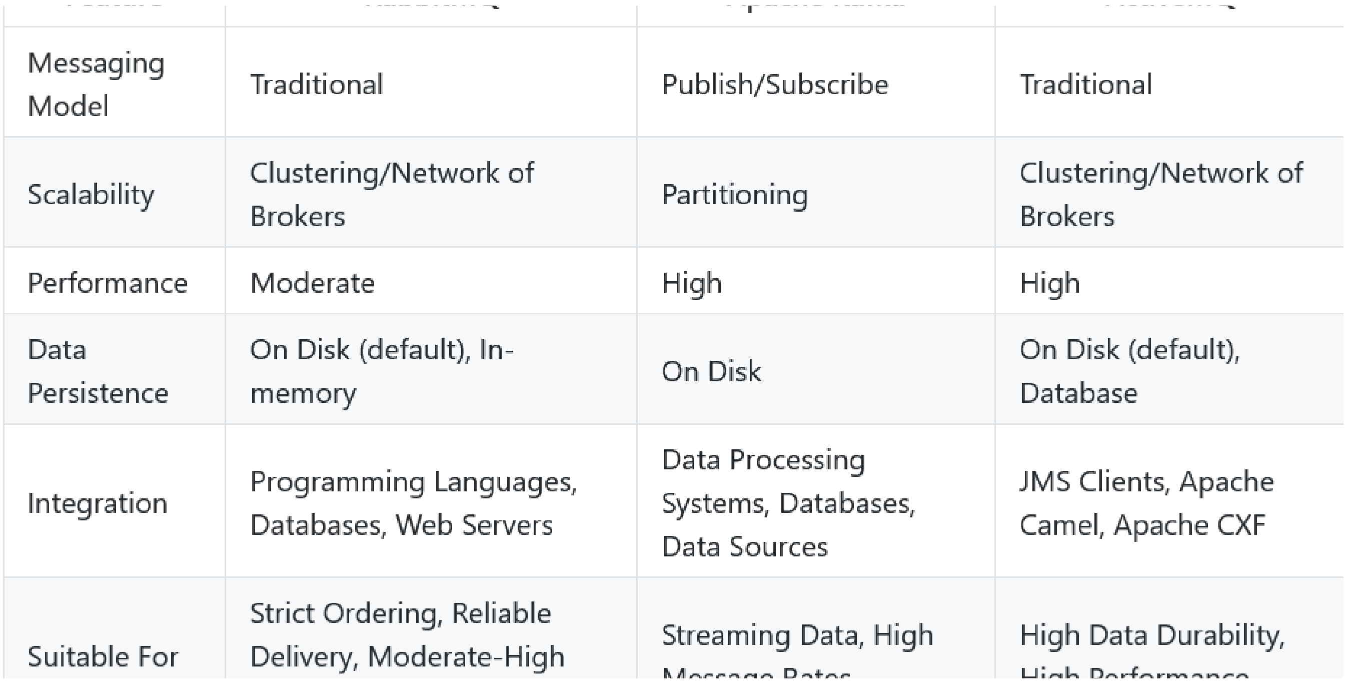
javinpaul in Javarevisited

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	High Data Durability,

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...

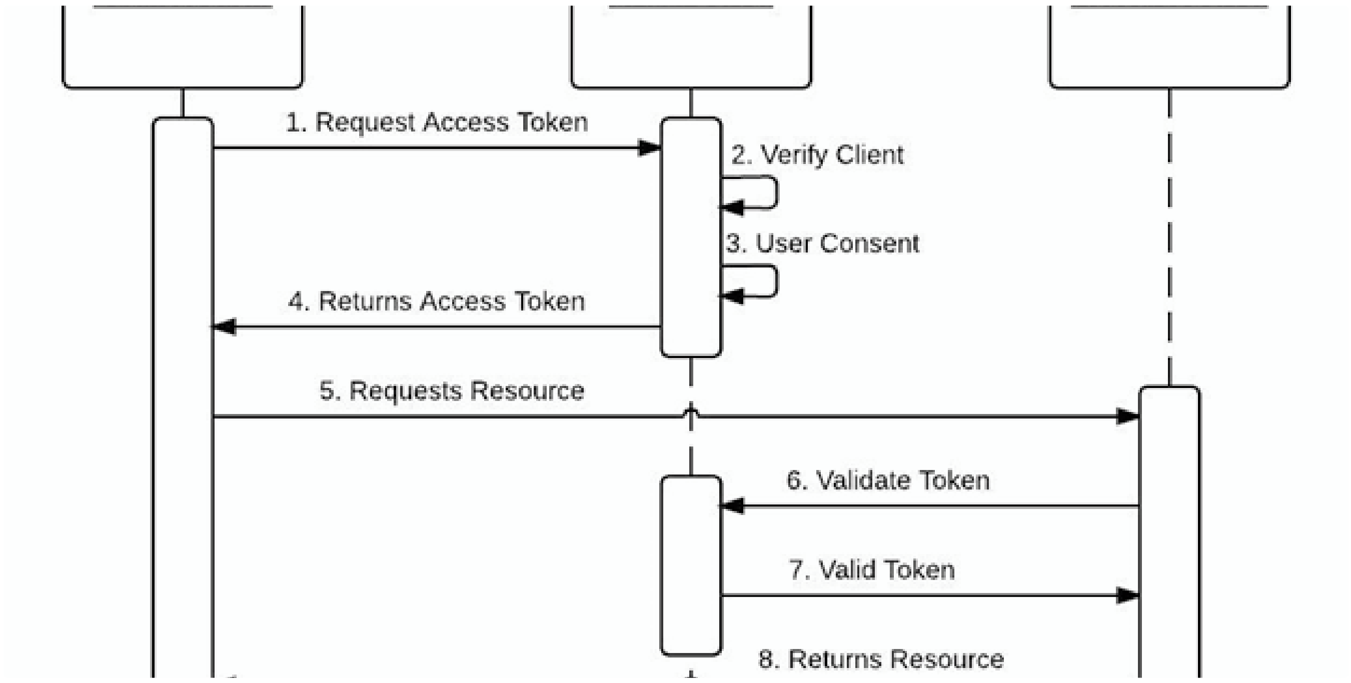
· 8 min read · May 19

208 4

See all from Soma

See all from Javarevisited

Recommended from Medium



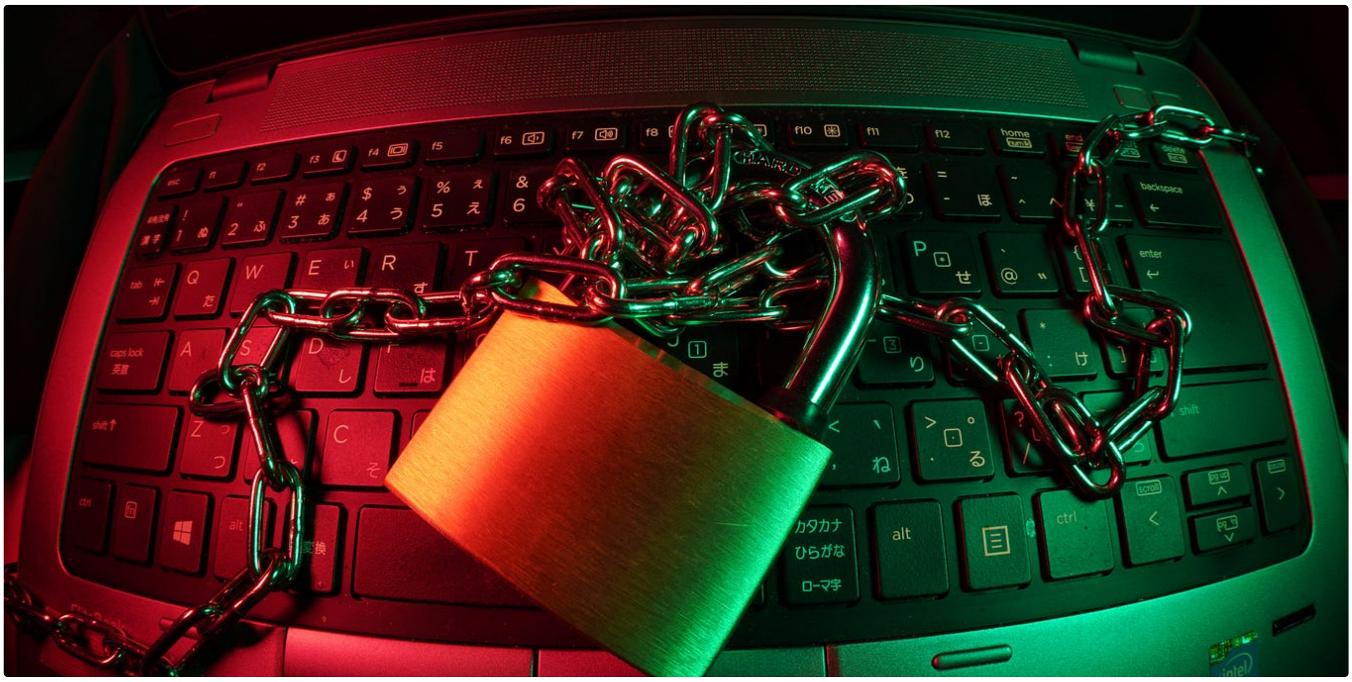
Somal Chakraborty in Dev Genius

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



Damith Neranjan in Geek Culture

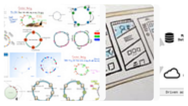
Securing Your Microservices Architecture with Spring Boot: An in-depth guide to Authentication and...

Microservices architecture has become a popular approach to building large-scale applications. It allows you to break down complex...

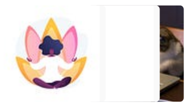
5 min read · Mar 24

30 1

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

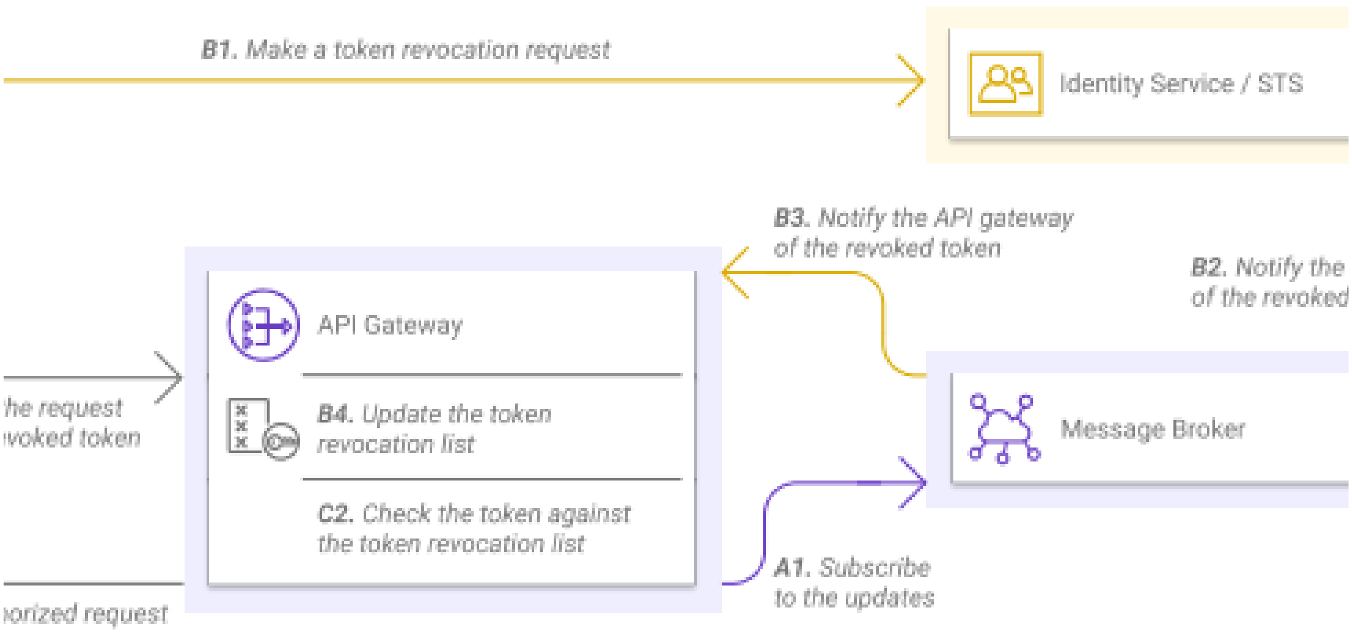
Amit Himani

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

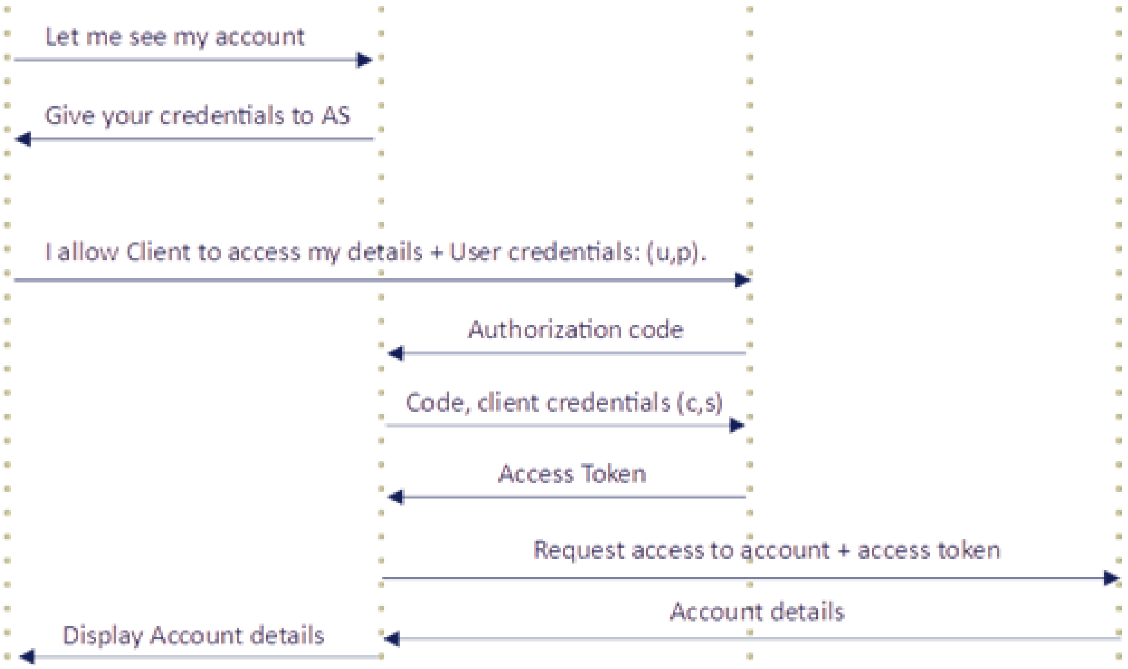

B1. Make a token revocation request
Identity Service / STS
B3. Notify the API gateway of the revoked token
B2. Notify the of the revoked
API Gateway
B4. Update the token revocation list
C2. Check the token against the token revocation list
he request voked token
Message Broker
A1. Subscribe to the updates
orized request

H Hamidrezakhaleghi

Architecture Approaches for API Gateway Access Control at Microservice

From the perspective of the end-user, accessing microservices via an API gateway is no different from accessing a monolithic application.

2 min read · 5 days ago


Let me see my account
Give your credentials to AS
I allow Client to access my details + User credentials: (u,p).
Authorization code
Code, client credentials (c,s)
Access Token
Request access to account + access token
Account details
Display Account details

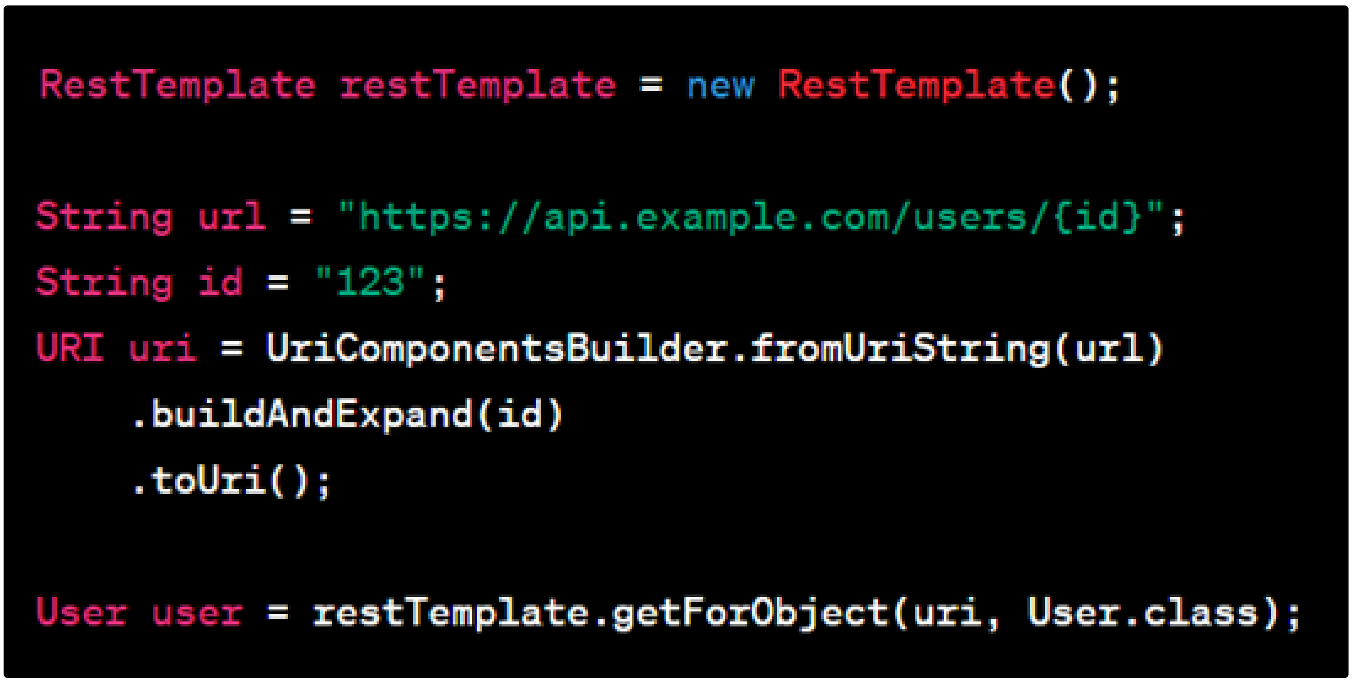
Miguel Pérez Sanchis in Julius Baer Engineering

Implementing OAuth2/OIDC and its agents with Spring

Intro

7 min read · Mar 23

61



```
RestTemplate restTemplate = new RestTemplate();

String url = "https://api.example.com/users/{id}";
String id = "123";
URI uri = UriComponentsBuilder.fromUriString(url)
    .buildAndExpand(id)
    .toUri();

User user = restTemplate.getForObject(uri, User.class);
```

Pavithra Radhakrishnan

Inter-Service Communication in Microservices

Java-based Web Service Clients Explained

3 min read · Jul 8

2

See more recommendations