Introduction to Microservices Security

Inside this section, we are going to focus on challenge 9 that we may face while building microservices.

I am pretty sure that I do not need to give any introduction about this new challenge.

Let me reveal what is this challenge about?

Like you can see here, this challenge is regarding **securing our microservices**.

I know you might be wondering throughout the course when this fellow is going to talk about the microservice security. To answer your burning question inside this section, I am going to completely focus on microservices security.

What a good web application or mobile application or microservices that you develop if you do not implement security inside them, then no one will recommend your application also to others.

Because without security, we will always expose sensitive data to everyone and that is going to be a problem for any organization.

So, to make you understand on why we should secure our microservices, let me ask some questions.

The very first question is **how are we going to secure our microservices from unauthorized users?**

As of now, anyone can invoke our microservices.

They can get the account details of anyone by providing their mobile number.

Do you think this is the right approach?

Off course not, because we are exposing everyone's data, even for the unauthorized users as well.

Apart from securing our microservices, we should also worry about how to enforce authentication and authorization inside our microservices.

There is a clear difference between authentication and authorization.

**Authentication** is a process of identifying the person or the application who is trying to invoke your

microservices.

Whereas **Authorization** which comes after authentication is a mechanism where we will enforce **privileged access**, which means the persons at the applications who has enough privileges, then only they will be authorized to access your microservices or web applications.

So, we should also worry about how to implement these authentication and authorization inside our microservices.

Apart from securing our microservices, we should also think about how to have a **single or centralized identity and access management component**.

If you are thinking like we can secure all our microservices individually, then I would recommend to

stop your thought there because we should never implement security logic in all the individual microservices, because that will be too much if you have hundreds of microservices, even if one requirement changes around the security, you will end up making changes in all the microservices.

That is why we should have a separate component which is going to responsible to store all the user credentials and handle authentication and authorization inside microservices network.

So, these are the three questions are the challenges that we need to think whenever we are trying to

secure our microservices.

Let me reveal what is the solution that we can follow to overcome all these challenges related to the

security.

With the help of security standards like **Oauth2 or OpenID Connect along with the Keycloak**,

which is an IAM product and **Spring Security, which is a spring provided project focusing on security**.

By using all these components, we can easily secure our microservices and handle all the above-mentioned challenges.

So, throughout this section we are going to leverage these products, frameworks and standards and we are going to implement security inside our microservices.

Spring security is a very fascinating concept.

*Inside this course I am going to talk about these on a very high level, but if you are new to the spring security framework and trying to learn the same, I would recommend to consider my course inside Udemy focusing on spring security.*

*Inside Udemy like you can see if you can search for spring security as of today, the very*

*first course that you can see in the results is my course, which is spring security six zero to master*

*along with the concepts like JWT tokens and Oauth2 framework.*

*Inside this course, I am going to talk about spring security framework from basics like I will assume*

*you do not know anything about spring security framework, and I am going to talk about everything about spring security framework. Like you can see as of now, this course has around 15 hours of content. I would strongly recommend you learning this spring security because this is a mandatory skill for any Java developer or a full stack developer. Without securing your web applications.*

*You cannot climb yourself as an architect or a senior developer. Like you can see,*

*this course as of now has around 29,000 students with a close to 4500 ratings, and this is one of the*

*best seller course. So, it is up to you. So, if you are interested, please enroll the same.*

*But like I said, we are going to use spring security framework inside this course as well.*

With this, I am assuming your super clear about the Challenge 9 that we are going to discuss inside

this section.

Problems that OAuth2 Solves

In the previous lecture, I told you that to secure our microservices, we need to use a component

called Oauth2.

**What is this Oauth2?**

Oauth2 is a security standard or a security specification that any organization can follow to secure

their web applications, mobile applications, or microservices, regardless of what is the type of

application that an organization develops, they can leverage these Oauth2 specifications to secure their applications.

Before I try to give you a quick introduction about what is Oauth2, first let me try to explain you,

**what are the problems that Oauth2 trying to solve?**

Because *before oauth2 specifications everyone they used to use* ***basic authentication***.

So, you may also have a question like why should we use Oauth2 framework for implementing security, why can't we use simple basic authentication?

To answer this, first, let me try to explain **what is basic authentication** and what are its drawbacks.

Back in the early days of web invention, all websites they usually ask for credentials of an end

user inside an HTML form. So, the end user will enter his credentials.

The same will be sent over to the backend server inside the backend server using the credentials provided by the end user, the authentication will be completed and after the successful authentication, the backend server is going to generate a session value and the same it is going to store inside a cookie of the browser.

So, if the session is active, the user can access the any protected resources and URLs.

So, this is how the basic authentication used to work.

The drawbacks of this approach are backend server will have both business logic and authentication logic tightly coupled.

**1st drawback:**

So, if there is a change that you need to make inside the authentication logic, then definitely you

need to make sure that it is not going to impact your business logic.

So, there must be enough regression testing has to be done.

Because both the business logic and the authentication logic is deployed into a single server.

Moreover, this approach is not mobile friendly or REST API friendly.

**2nd drawback:**

And the second drawback of basic authentication is, it does not accommodate or it does not have a proper solution to grant temporary access to your information to the third-party clients. For example, think like I am storing my photos inside Google photos and one of the third-party websites

they offer very good features, like if I can import my photos from Google photos into this website,

they will allow me to edit the photos to prepare a collage from the photos so I can do many inside this third-party website.

But like you can see here, there is a challenge how to bring the photos that I am storing inside Google photos into this website.

If we follow basic authentication approach, then we need to provide our own Google credentials to this website.

We need to believe this website folks and we need to provide our own credentials under the assumption that they are not going to misuse our credentials.

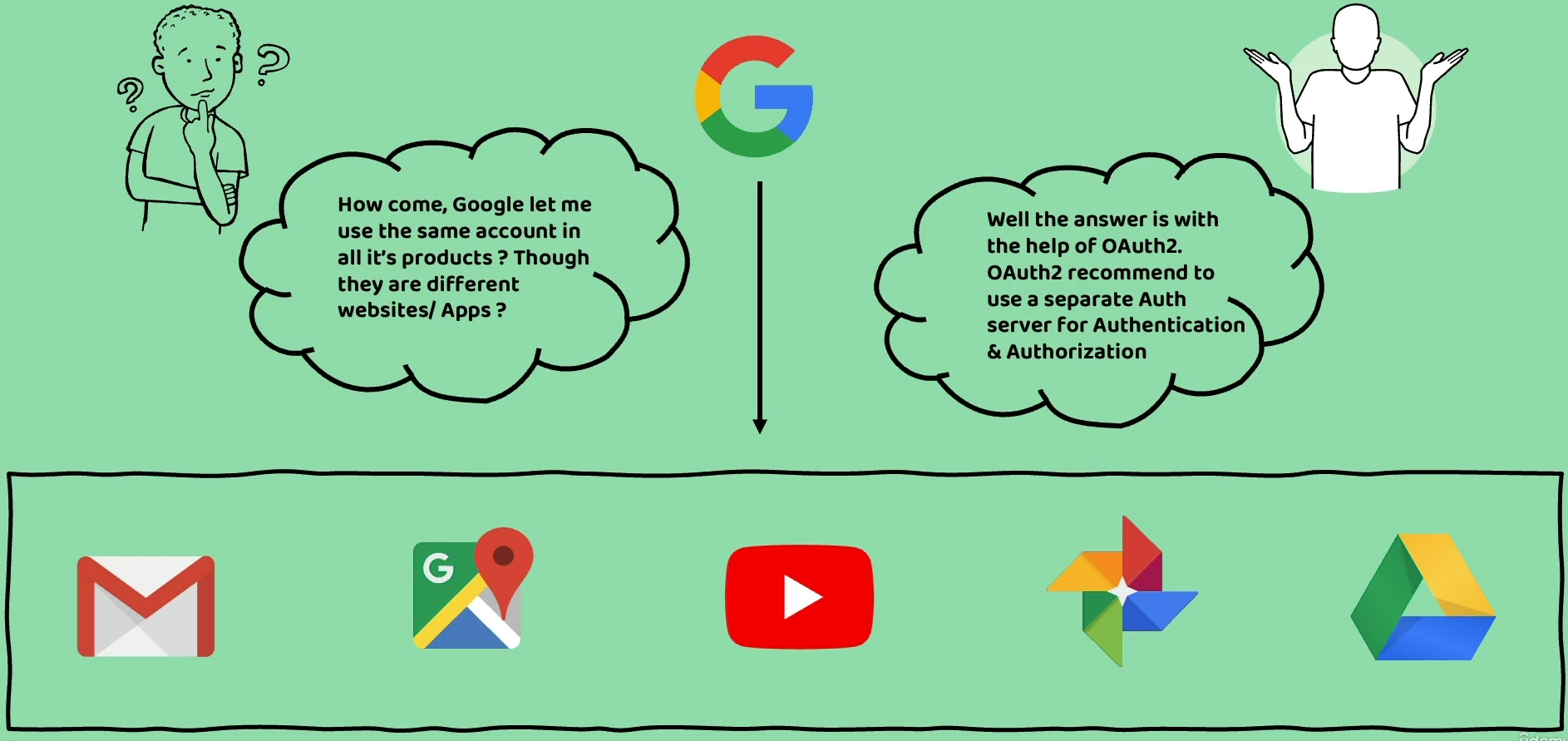
So, this is a very serious drawback.

Whereas in Oauth2 framework, we have a proper solution on how to temporarily grant access to the Google photos to the third-party website without sharing my own Google photo credentials, I am going to talk more about these in the coming lectures.

But as of now I am assuming you are clear about why basic authentication is not a good consideration these days and what are the challenges that we have.

Now let us try to understand what are the problems that Oauth2 trying to solve.

**1st Problem solved:**

****

We all know that Google has many products like Gmail, Google Maps, YouTube, photos, Google Drive.

So, under Google there are many web applications or mobile applications available providing various features to the end users.

Have you ever wondered like, how come Google let me use the same account in all its products?

Regardless of what is the website of Google that you are using or what is the mobile app of Google

that you are trying to use.

If I go to the Gmail and try to login, I need to enter my Gmail credentials.

If I try to use another product of Google, like YouTube or Google photos or Google Maps there also

I need to enter the same credentials which I have used for my Gmail login.

Do you know how this is possible?

This is possible because behind the scenes, **Google uses the Oauth2 framework** and by following the Oauth2 recommendation, they separated all the authentication and authorization logic into a separate component called **authorization server** or **authentication server**.

So, whenever an end user trying to login into any of the Google products, they are going to send the

credentials of the end user to the same auth server.

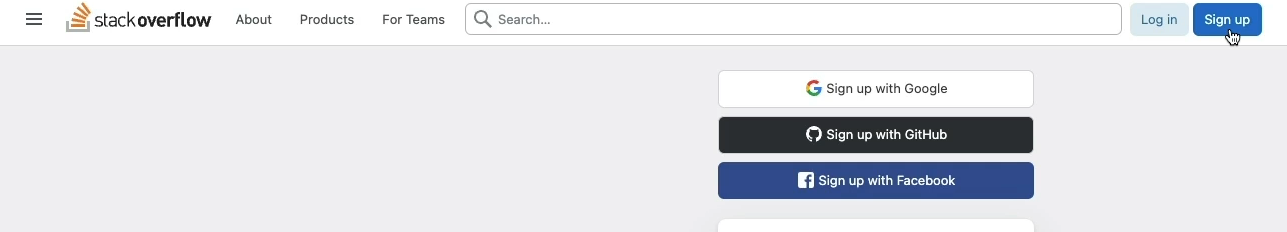
So, from this, I am assuming it is clear for you one of the best recommendations of Oauth2 framework is, to separate all your authentication and authorization logic into a separate component.

So that regardless of how many applications you have or how many microservices that you have inside your organization, you can always have your security related logic inside a single component.

And if there is a change that you need to make, then the change is going to be in a single place.

So, this is the very first problem that Oauth2 is trying to solve.

**2nd Problem solved:**



Now let us try to understand what is the second problem that Oauth2 is trying to solve. For the same, I am going to give you a small demo.

I came to our favourite website, which is Stack Overflow.

Like, you know, Stack Overflow is all the developer's favourite website.

Inside this website, let us assume you do not have any account as of now.

You are trying to use this website where you are trying to post a question. For the same, you are going to click on this sign-up button.

As soon as you click on the sign-up button, you can see there are multiple approaches on how I can

create an account inside this website.

The most basic approach is I can enter my name, what is the email, what is the password.

And with that an account will be created behind the scenes in the Stack Overflow.

But apart from this approach, I can also sign up with Facebook, GitHub, and Google.

So here you may have a question like how the Stack Overflow will know, what are the credentials of my Google, GitHub, or Facebook.

The answer is, Stack Overflow does not need to know your actual credentials.

With the help of Oauth2 framework, they can get your basic details like what is your name,

what is your email.

Now my Stack Overflow has my basic details like this is my email, this is my display name, which I have stored inside the GitHub. Which means the resources that I stored inside the GitHub server now is shared to the Stack Overflow with the help of the Oauth2 framework.

And in the process, I did not share my credentials to the Stack Overflow.

I entered my own credentials to the GitHub and behind the scenes GitHub gave my basic details to the Stack Overflow and it also going to issue an access token and using this access token in future, maybe after two days or three days, if I came back into this website, the behind the scenes is that Stack Overflow can try to send a request to the GitHub and try to log in me automatically with the help of the access token.

So, what is happening here is, the GitHub is trying to give a temporary access token with a limited access to the Stack Overflow.

**I did not share my master credentials to the Stack Overflow and at the same time GitHub also did not give an access token using which the Stack Overflow can do anything on my GitHub.**

**It has given its access token, which has a very limited privileges** like my stack overflow can read what is my name, what is my email, that is it.

Apart from reading these details, the Stack Overflow website cannot perform any other operation inside my resources in the GitHub.

It cannot create repositories.

So those are advanced privileges which GitHub did not give.

It is only given a very basic privilege which is reading the profile details like email and my display

name.

So, this is one of the problems that Oauth2 is trying to solve.

Without sharing our credentials to the third-party applications, we can still provide a temporary access to them to access the resources that I stored inside a resource server like GitHub, Google or any other application which has implemented Oauth2 inside their system.

I hope now you are clear.

What is basic authentication, what are its drawbacks and what are the problems that Oauth2 is trying to solve.

Introduction to OAuth2

Inside this lecture, let me try to give a quick introduction about what is oauth2.

**OAuth stands for Open Authorization.**

It is a free and open-source protocol built on IETF standards, and the license also is provided from

the Open Web Foundation. That means anyone can use these specifications inside the OAuth framework. The current version of OAuth framework is the second version. That is why many people call this as Oauth2. That means they are trying to refer to the second version of OAuth. And right now, the authors of OAuth2 are trying to build a newer version with the name OAuth 2.1.

So, like we discussed previously, with the help of these or 2 or 2.1, an application can grant permissions to access your data in another third-party application.

**The process of granting permissions is concerned to third party applications are often referred to**

**as authorization or delegated authorization**.

Because inside this process ***you are going to authorize one application to access your data in another application on your behalf without giving them your password***.

That is why, in another words, we can also call this process as **delegated authorization**.

So, this is a very basic definition of OAuth2.

Now coming to the major advantages of Oauth2.

OAuth2 supports all kinds of applications and scenarios that you can imagine in the web world,

regardless of what type of applications you are building.

For all these applications and scenarios, there are different, **different grant flows available inside**

**the Oauth2 framework**.

Suppose if two backend servers are trying to communicate, then we have a grant flow specific to that

scenario.

Whereas a UI application or a mobile application is trying to communicate with a backend server, then there is a separate grant flow for it.

And very similarly for IoT devices, consoles, smart TVs.

So, for all these kinds of devices also we have a grant flow inside the Oauth2 framework.

We are going to discuss few of them in this section.

But on a high level, what I am trying to say here is, regardless of what is the type of communication

that you are trying to establish between two different applications or two different devices.

The OAuth2 has answer for all of them with various grant flows.

And the next advantage of OAuth2 like we discussed, it recommends separation of auth logic. Inside

OAuth2 we have to build a separate component with the name authorization server and this **authorization server is responsible to receive the request from the clients and provide access token based upon the successful authentication**.

Regardless of how many applications or what type of applications you have inside your organization,

they can all connect to the Auth server to perform the login operation.

Not only the own organization applications, but also the Auth server can also be accessed by the third party web applications.

Inside the Stack overflow scenario, we saw that it was able to access the Auth server of GitHub and it

can authenticate an end user and get the details of the end user with the help of GitHub auth

server.

This is possible because inside OAuth2 we have a separate server which is responsible purely for authentication and authorization.

If you club both authorization and business logic into a single server, then organizations will not

dare to expose their server details or endpoint details to the other applications.

So, all the credentials of the user and client applications will be maintained in a single location which

is inside the server.

And the next advantage, which is a primary advantage of OAuth2 like we discussed.

The end user, he does not have to share his credentials whenever he is trying to provide access to his data to a third-party application.

The auth server which is present inside the OAuth framework.

It is going to provide a temporary access token to the third-party application.

*This is very similar to like a hotel scenario.*

*Think like you went to a five-star hotel and you are trying to check in into that hotel. At the reception*

*once they validated your booking details, they are going to provide an access card.*

*And using this access card, you can only open your own room and you can only take the elevator to the floor where your room is present.*

*Here the hotel people they are trying to give a temporary access card and even if you lose the access*

*card, it is not going to create any problem because the hotel people can invalidate that access card*

*remotely and they can provide you a new access card.*

Very similarly inside the OAuth2 framework also, the auth server is going to issue the access tokens based upon the privileges. Just like inside the hotel scenario, the access card that you hold will only provide access to your own room, whereas the access card given to the housekeeping department.

They can access any room because the privilege for that access card is more this way based upon the

business requirements.

The OAuth server is going to issue different, different access tokens based upon the privileges needed by the end user. I hope you are clear with the introduction of Oauth2.

OAuth2 jargons or terminologies or roles

What is OpenID Connect and Why it is important

Introduction to IAM products and why Key Cloak

Deep Dive of Client Credentials grant type flow

Securing Gateway server using Client Credentials grant type flow – Theory

Setup Auth Server using Key Cloak

Register client details inside Key Cloak for Client credentials grant flow

Getting Access Token from Auth Server in Client credentials grant flow

Securing gateway Server as Resource server – Part 1

Securing gateway Server as Resource server – Part 2

Implement Authorization inside Gateway server using Roles – Part 1

Implement Authorization inside Gateway server using Roles – Part 2

Deep Dive of Authorization Code grant type flow

Securing Gateway server using Authorization Code grant type flow - Theory

Register client & end user inside Key Cloak for Authorization code grant flow

Demo of Authorization code grant type flow

Demo of Microservices Security using Docker containers & Docker compose - Part 1

Demo of Microservices Security using Docker containers & Docker compose - Part 2