Microservices authentication

In this lesson you will learn about the need to introduce identity related components to your microservices based application, the dangers of not doing this correctly and the modern approach to securing your services.

The need for authentication

Let's go once again through one of the scenarios enabled up to this point. John, one of our players, would like to see the items in his inventory bag. He would request his inventory via his preferred client, and this one in turn would send a GET request to the Inventory microservice asking for all items associated to the specified userid.

Normally, the Inventory microservice would just return the items, however our service should probably start asking a few questions before handing them over. For instance, Inventory may want to ask "Who are you?" or "Are you allowed to talk to me?", since it should probably not just hand over its precious data to anybody that asks for it. But even if the request is allowed to come in, another important question arises: "Do you have access to John's inventory bag?".

Think about it, Inventory doesn't really know that the caller is John, it could be any other player trying to retrieve John's inventory, which by all means should not be allowed. Only if our microservice can answer these questions, and can confirm that the caller has access to the items, it would query them from its database and return them.

In summary, we need to establish the identity of the caller.

Adding authentication

The good thing is that we do have an Identity microservice by now, so we should be able to enable authentication in our system, that is, we should be able to confirm that the caller is allowed to retrieve the items for the specified user.

For instance, the client could be modified to present a sign in screen to collect the email and password of the user and then send them in the request to the Inventory microservice. Inventory would then ask Identity to confirm the credentials are valid and probably would also ask to return additional info about the user, like its user id, to correlate with the Inventory database. If all looks correct, Inventory would be ready to return the data to the caller. We would also update our other microservices, like Catalog, to receive the credentials and do the corresponding checks with Identity.

The problem with this approach is that we would be sending the user's credentials on every single request to each of our microservices, which presents a few concerns:

- If SSL is not enabled, which of course should not be the case, an attacker with a package sniffer could easily grab the credentials
- Even if SSL is enabled, tools like Fiddler, capable of decrypting the URLs, could have been installed in the users machine, assuming the user provided consent for it
- The URLs will likely get logged on each microservice, and the logs might not be secured properly

- The size of each request would be bigger than it needs to be and there are limits to the request URLs
- The client would need to ask for credentials every time or perhaps would keep them in its local cache, which is just as bad
- And let's not forget that we might overwhelm our Identity microservice with potentially hundreds of checks on every request from each of our microservices, which will of course also slow down the overall system unnecessarily

There's probably a better approach to authenticate our users.

Giving away your credentials

As a side note, I just wanted to call out a bad practice of some web sites that you might be familiar with. Have you ever tried using one of these sites that ask for your bank account credentials so it can help you manage your finances? Think about what these web sites are doing, they would even assure you your data is very safe since "your login username and passwords are stored securely in a separate database using multi-layered hardware and software encryption".

Even with their best intentions this is definitely a bad idea because nobody other than your bank, who owns your credentials, should have a copy of them, ever. You really don't know what this third party is planning to do with your credentials, how safely it is storing them and also what exactly you are allowing them to do with your bank account. Presumably this is read only access to your account, but how can you be sure?

We won't be dealing with external web sites in this course, but the appropriate authentication and authorization patterns you will learn next still apply for any scenarios like this.

Enter token-based security

What is then the right way for our client to request access to John's inventory bag? The ideal and modern approach is called token-based security, and it works like this:

When John wants to get his inventory bag, the client first requests authorization from an authorization server, which in our case would be our Identity microservice. It is this microservice, not the client, that will then present John with the sign in form, where he will enter his username and password.

Once John submits this form, his credentials will be sent to the Identity microservice, not to the Inventory microservice, for validation. Once Identity has verified the credentials it generates a signed authorization token and sends it back to the client.

The client can then use this authorization token to try to access John's inventory bag in the Inventory microservice. If Inventory can verify that this token presents the required authorization, it will go ahead and return the items to the client.

Notice that with token-based security the credentials are sent only once from the form presented by the authorization server, and not on every subsequent request. There is more going on here before generating the token, but we will look into these details in the next lesson.

The generated token represents consent from the user and it includes information about:

- How to verify the token, since it includes details about how it was signed
- Where it came from, since only tokens generated by our authorization server should be allowed
- Where it can be used, because it should be valid for use only in our microservices
- Who is authorized by the token, since it can include many details about John in this case, including his authorization roles
- What can be done with the token, because the token may grant access only to some microservices, or even to specific actions within a microservice
- An much more

Now the question is, how to generate this token and how to do it in a standard way so that we follow all the best security practices while not having to spend a lot of time writing code to get things right? Luckily, there are already well-known protocols and standards that we can use for this.

In the next lesson we will learn about OAuth 2.0, the de facto standard for authorization these days.