Xavier Hahn · Follow
Nov 13, 2017 · 5 min read · Listen

Adding authorization to Asp.Net Core app using Keycloak

Using Keycloak's roles functionalities to setup authorization in Asp.Net Core web apps and APIs.

This is a follow-up to my previous story where I explained how to setup an Asp.Net Core app and an Angular app to authenticate users through a Keycloak server using OpenID Connect. In this article, we will go a step further and add authorization using Keycloak's built-in "roles".

Authorization (geralt from pixabay)

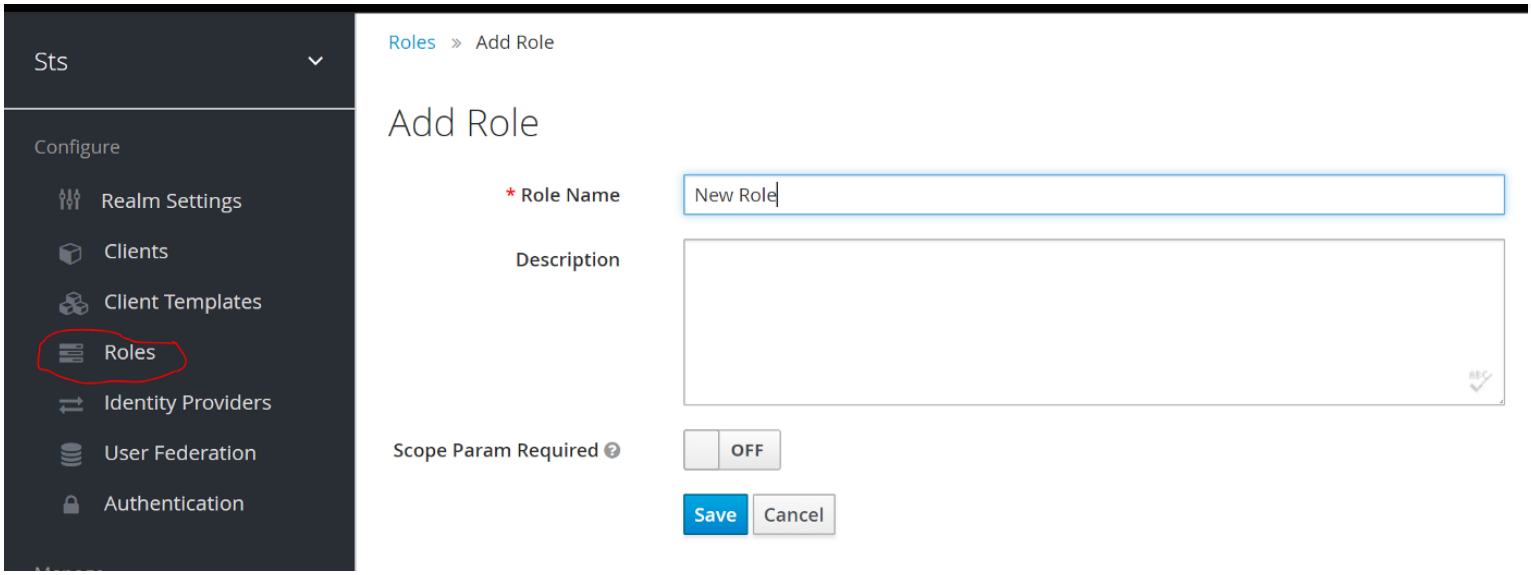
Setting up Keycloak

I won't go into all the details about how to setup a Keycloak server in this article as this was covered extensively in the previous article. Therefore, you should have a Keycloak server running, with a few users setup and a client app configured for your Asp.Net Core Api.

Keycloak's roles

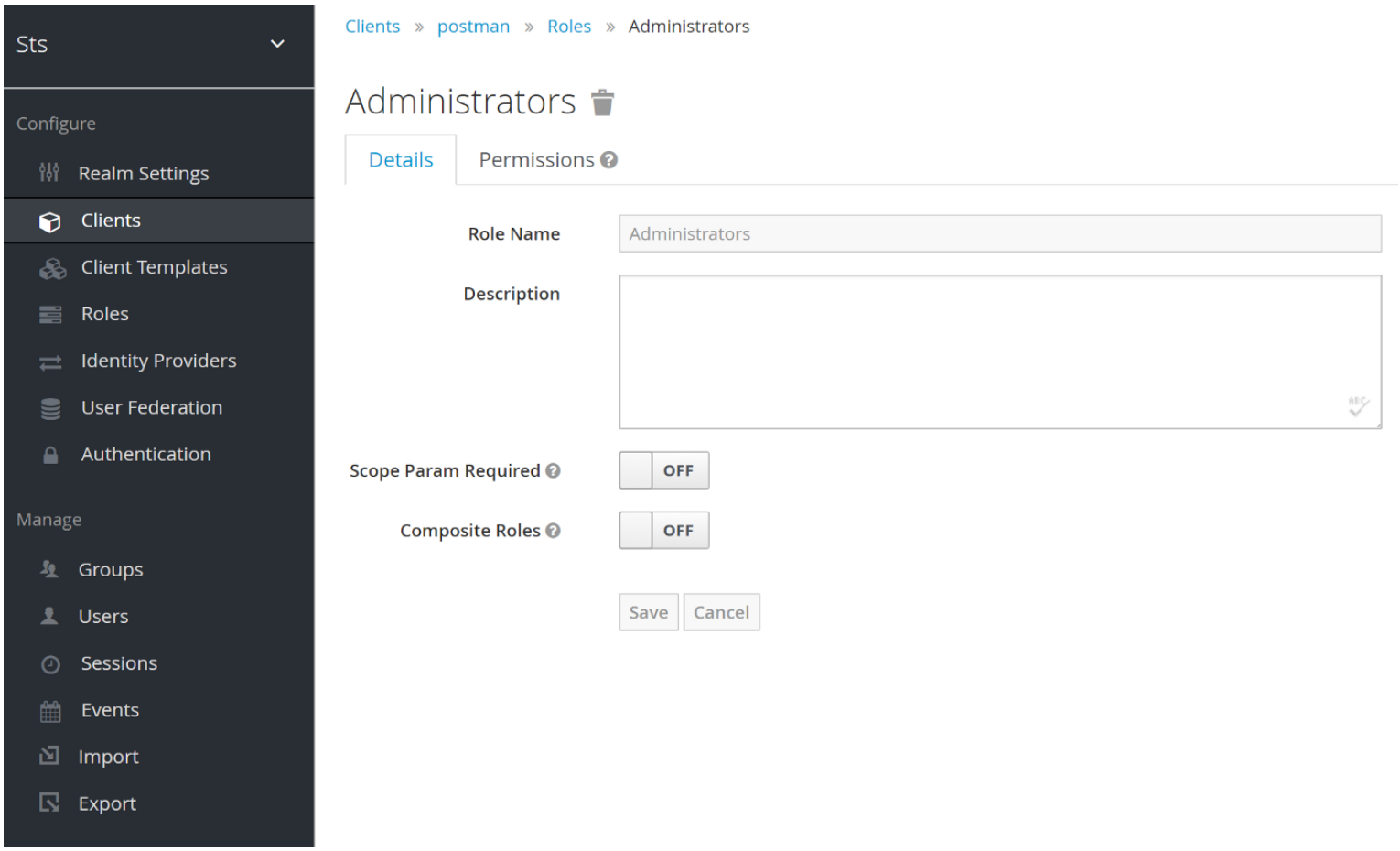
There are two types of roles in Keycloak, *Realm roles* and *Application roles*. Realm roles are roles that are setup at "Realm" level, this is useful if you need to "share" roles across applications boundaries. For example, if you have roles that are global to the company, that's where you'll want to add them.

To add realm roles, simply connect to Keycloak's administrative console, select the role for which you want to setup roles and click on the *Roles* tab. It will list the current roles available in the realm, click on the *Add role* button. Add the role name, select save and you will have your new role.



Adding a new role

Clients roles, are roles that are available only for one specific client application. This is a very likely scenario, as users that are administrators for one application might be a simple users for another. To add client roles, you need to select the *Clients* menu, then select the client for which you want to add roles. Select the client that we have created for our application in the previous article. Then, select the *Roles* tab, you should see an empty list. Click *Add role* and add a role named *Administrators*. When you save, you should get to the newly created role details.



Adding a role to a user

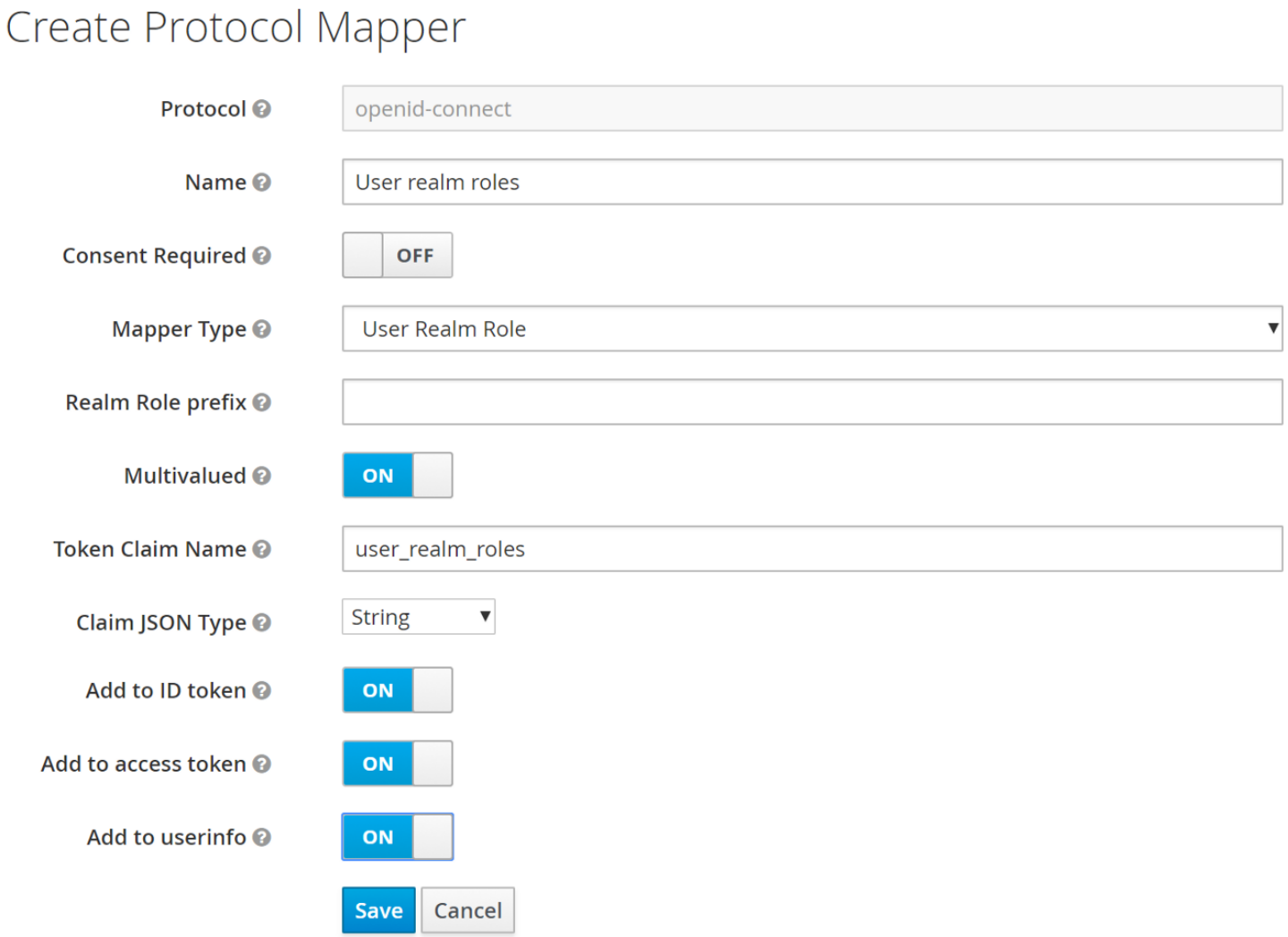
Now, let's get to the *Users* menu and map a role to a user. In the users list, search for a user and click on its *id* to get to its details page. From there, select the *Role mapping* tab and you should see the realm role that you created at the top. Select the client in the drop down list and you should see the *Administrators* role that you created earlier.

Add both roles to the user.

see a number of built-in mappers.

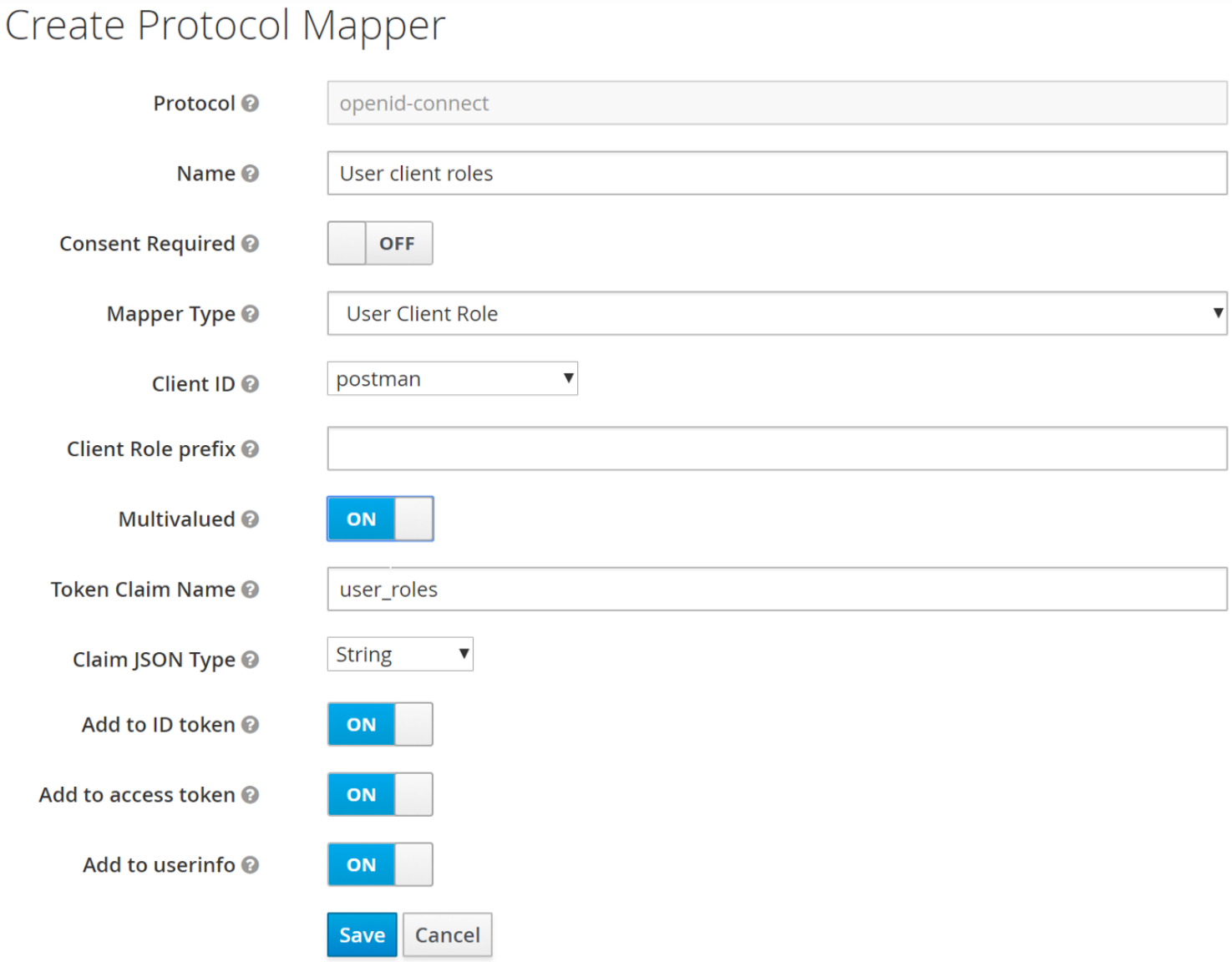
Click on the *Create* button and fill-in the form as follow:

- **Name:** can be anything you want, I'm using "User realm roles"
- **Mapper Type:** User Realm Roles
- **Multivalued:** true
- **Token Claim Name:** again, anything you want, I use user_realm_roles
- **Claim JSON Type:** string
- **Add to ID token, add to access token, add to user info:** true



Create another mapper, this time for the client role.

- **Name:** can be anything you want, I'm using "User client roles"
- **Mapper Type:** User Client Roles
- **Client ID:** the client id of your application
- **Multivalued:** true
- **Token Claim Name:** again, anything you want, I use user_roles
- **Claim JSON Type:** string



> *Remark: here I'm creating both a client and a realm role, but this is not mandatory, depending on your application, you could have either only client roles, only realm roles, or both.*

Testing the configuration

We can do a little test using Postman and jwt.io to check that our mappings are correctly sent in the claims.

Open Postman, select the *Authorization* tab of any request and click on the *Get New Access Token* button. Enter a token name, and select *Authorization code* for *Grant Type*. Fill in the rest of the form as follow:

- **Callback URL:** Anything, but make sure that you have added it to the list of allowed URLs for the client in Keycloak
- **Auth URL:** http://localhost:8080/auth/realms/master/protocol/openid-connect/auth
- **Access Token URL:** http://localhost:8080/auth/realms/master/protocol/openid-connect/token
- **Client ID:** The client ID of your application

The rest of the fields can be left as default. Click *Request Token* and authenticate with the user for which you added the roles. This will save your access token. Copy it and paste it to jwt.io. This will decrypt the token, allowing you to inspect the content. You should have both the *user_realm_roles* and *user_roles*. Great, we're now ready to add the authorization to the Asp.Net Core Web API.

Adding authorization to Asp.Net Core Web API

We'll start from the .Net Core solution as we left it at the end of the previous article. We're going to use Asp.Net Core's built-in Role Based

Open the *Startup.cs* file and add the following lines after the `AddAuthentication` call in the `ConfigureServices` method.

```
services.AddAuthorization(options =>
{
  options.AddPolicy("Administrator", policy => policy.RequireClaim("user_roles", "[Administrator]"));
});
```

startup.cs hosted with ❤ by **GitHub** **view raw**

Now, we can simply add the authorization attribute with the role property set to Administrators to the controller classes or methods that we want to protect.

```
[HttpGet("authorization")]
[Authorize(Roles = "Administrators")]
public string TestAuthorization()
{
    return "Authorization is working!";
}
```

testcontroller.cs hosted with ❤ by **GitHub** **view raw**

Run the project and use Postman to test your application with the authorization working correctly. Only users with the administrators role should be allowed to access methods.

> *Remark: As you can see, the RequireClaim doesn't seem to understand that the claim is an array, and treats it as a string instead, which is why you need the [] characters. This means that it won't really work for cases where the user has more than one role. I don't have a definitive answer as how to fix that issue. The only option I see for now is implementing a custom policy-based authorization.*

Conclusion

As you can see, adding authorization to an Asp.Net Core Web API from Keycloak's roles is relatively easy. Since Keycloak allows both client and realms-type roles, it offers a wide range of possibilities.

I've updated the previous article's source code to add the authorization example from this article.

Thanks for reading this article, feel free to add your comments, remarks or question and to share this article.