

Keycloak in Docker #4 – How to define user privileges and roles

📅 25th December 2021 / 👤 little_pinecone / 📁 Tools

Keycloak offers a wide variety of methods for defining user permissions and roles. We can configure privileges across a realm or a specific client application. In addition, we can combine permissions by assigning users to groups or creating composite roles.

Prerequisites

- I'm going to use the [keycloak-in-docker](#) project to run an example dockerized Keycloak server. You can learn how to build a service like this in the [Keycloak in Docker #1 – How to run Keycloak in a Docker container](#) post.
- My example **keycloak** container automatically imports a default realm with users. You can learn how to import a realm in the [Keycloak in Docker #2 – How to import a Keycloak realm](#) post.
- The Keycloak server will be available under the <http://localhost:8024/auth/> url. The admin credentials are **keycloak:keycloak**.
- The imported users have a **user** realm role by default.

User roles in Keycloak

Role is an abstraction that helps to incorporate access details and privileges into a convenient concept. Thanks to this, we can easily assign a set of permissions to users, user groups or more complex roles.

Keycloak roles are defined in a dedicated namespace so that all users with the same roles have identical permissions in that namespace. In other words, realm-level roles are a global namespace for a given realm, while client roles are namespaces intended for specific applications.



- Composite roles.

What's more, we can combine these approaches to fine-tune permission management.

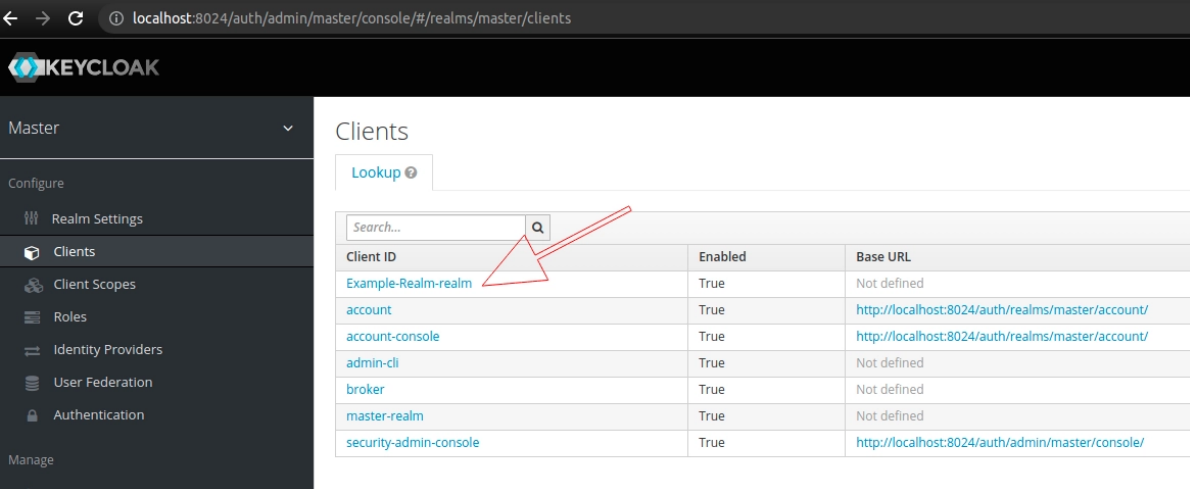
Master realm roles

Master is a unique realm that is available by default to allow admins to control other realms. Here we can create multiple super users who will be able to manage all or only selected realms. The documentation specifies two default realm-level roles for master: **admin** and **create-realm**:

Users with the **admin** role are super users and have full access to manage any realm on the server. Users with the **create-realm** role are allowed to create new realms. They will be granted full access to any new realm they create.

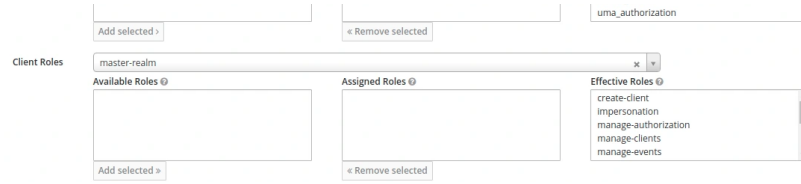
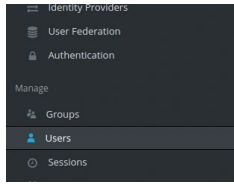
https://www.keycloak.org/docs/16.0/server_admin/#global-roles

Master views other realms as clients with the **-realm** suffix appended to their names. Consequently, in our example Keycloak server, the **Example-Realm-realm** is listed among the clients of the master realm:



Client ID	Enabled	Base URL
Example-Realm-realm	True	Not defined
account	True	http://localhost:8024/auth/realms/master/account/
account-console	True	http://localhost:8024/auth/realms/master/account/
admin-cli	True	Not defined
broker	True	Not defined
master-realm	True	Not defined
security-admin-console	True	http://localhost:8024/auth/admin/master/console/

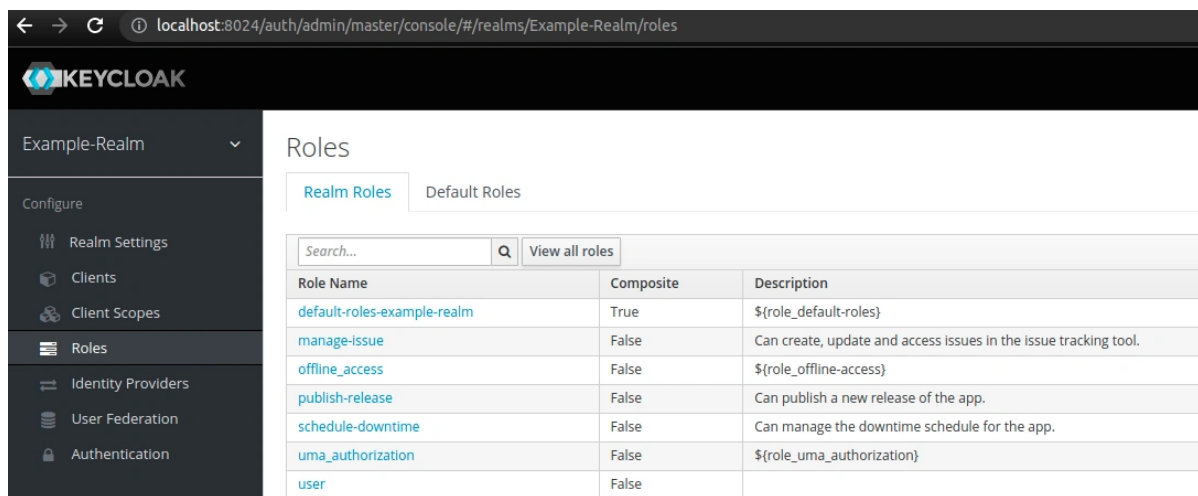
Furthermore, we can check the role mappings for our default admin account:


 Search ...


Additionally, we can find more details and specifications of the experimental features in the [Master realm access control](#) section of the Keycloak documentation.

Realm roles

We need to select **Example-Realm** to view all roles available in it. In addition to the default **offline_access** and **uma_authorization** roles, we can see a role that I created specifically for this realm – **user**:



Role Name	Composite	Description
default-roles-example-realm	True	\$(role_default-roles)
manage-issue	False	Can create, update and access issues in the issue tracking tool.
offline_access	False	\$(role_offline-access)
publish-release	False	Can publish a new release of the app.
schedule-downtime	False	Can manage the downtime schedule for the app.
uma_authorization	False	\$(role_uma_authorization)
user	False	

In order to see users associated with a given role, click the role name and go to the **Users in Role** tab:

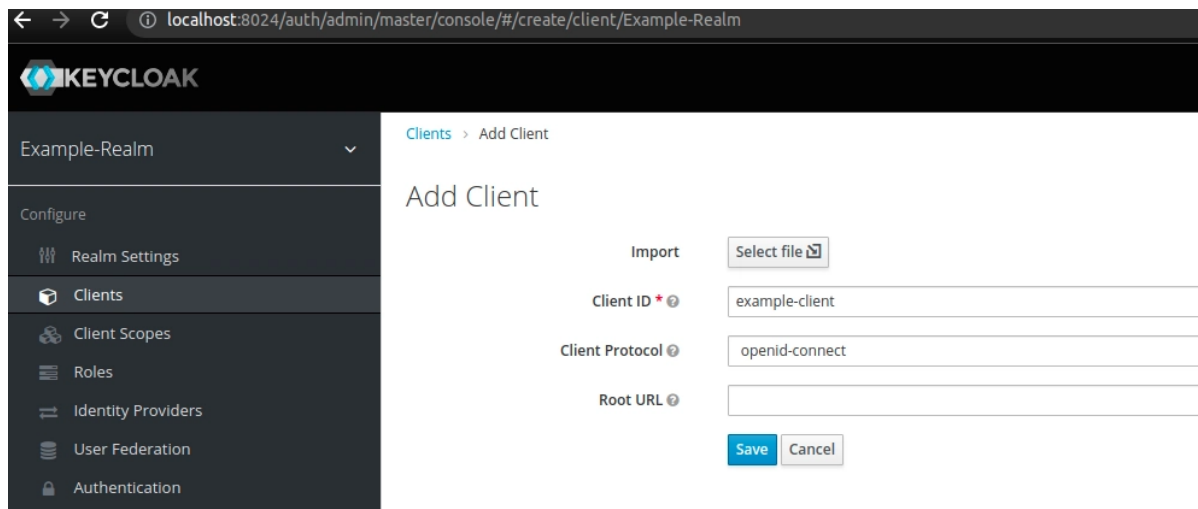


Username	Last Name	First Name	Email
hanna	Davis	Hanna	hanna@test.com
carlo	Velazquez	Carlo	carlo@test.com
noel	Horton	Noel	noel@test.com
christina	Travis	Christina	christina@test.com

The **roles defined within a realm** will be effective across all its clients. As a result, we can use them to provide more coarse-grained access control.

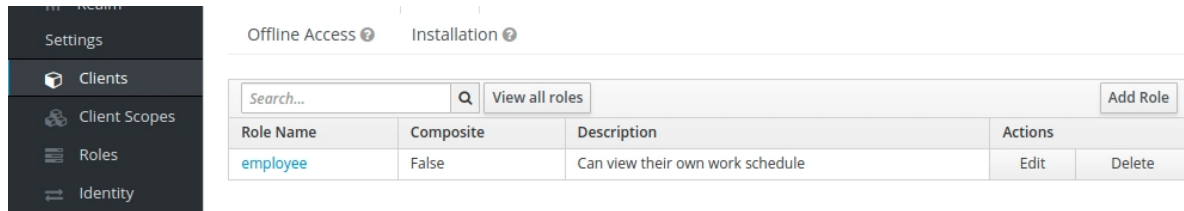
Client roles

In Keycloak, each **client gets its own role namespace** to hold user roles. Roles defined in one client are invisible to other clients. First, we're going to create a sample application for our realm:



The screenshot shows the Keycloak Admin Console interface. The left sidebar contains a menu with options: Clients, Client Scopes, Roles, Identity Providers, User Federation, and Authentication. The 'Clients' tab is selected. The main area displays the 'Add Client' form. The form includes fields for 'Client ID' (set to 'example-client'), 'Client Protocol' (set to 'openid-connect'), and 'Root URL'. There is an 'Import' button with a 'Select file' dropdown. At the bottom of the form are 'Save' and 'Cancel' buttons.

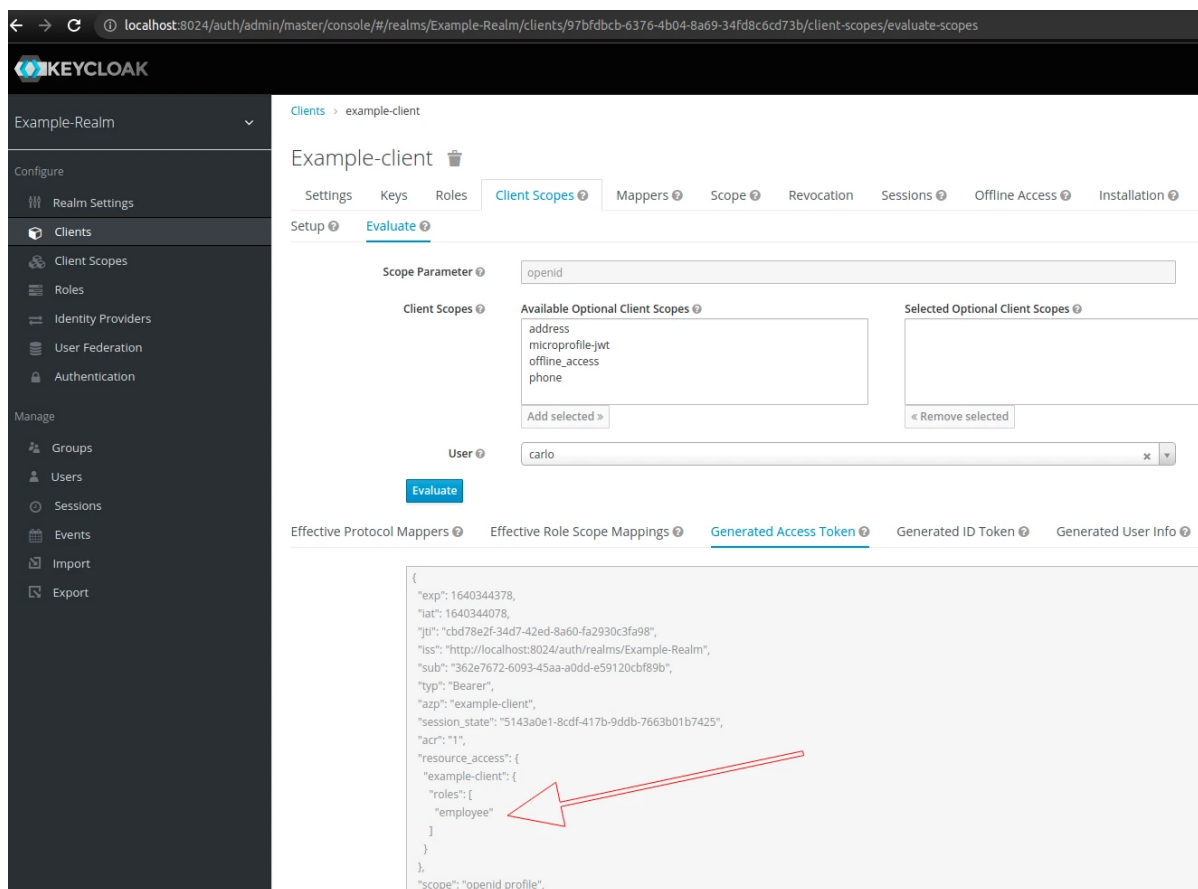
Then, we can select the **Roles** tab and click the **Add Role** button to create the **employee** role. Finally, we can see all roles assigned to that particular client, just like on the screenshot below:



Role Name	Composite	Description	Actions
employee	False	Can view their own work schedule	Edit Delete

Now we can assign this role to users. Keep in mind that this role is only effective for that particular client. It won't be added to the realm namespace by default, nor will it be visible for other clients.

To see users with a specific role, click the role name and select the **Users in Role** tab. In addition, we can also preview what client roles will be included in the access token for a given user. Go to **Clients** → **Client Scopes** → **Evaluate**, select a user, click the **Evaluate** button and go to the **Generated Access Token** tab, as shown on the screenshot below:



```
{
  "exp": 1640344378,
  "iat": 1640344078,
  "jti": "cbd78e2f-34d7-42ed-8a60-fa2990c3fa98",
  "iss": "http://localhost:8024/auth/realms/Example-Realm",
  "sub": "362e7672-6093-45aa-a0dd-e59120cbf89b",
  "typ": "Bearer",
  "azp": "example-client",
  "session_state": "5143a0e1-8cdf-417b-9ddb-7663b01b7425",
  "acr": "1",
  "resource_access": {
    "example-client": {
      "roles": [
        "employee"
      ]
    }
  },
  "scope": "openid profile",
}
```

The client roles are listed in the **resource_access** section of the token.



Group hierarchy and role inheritance

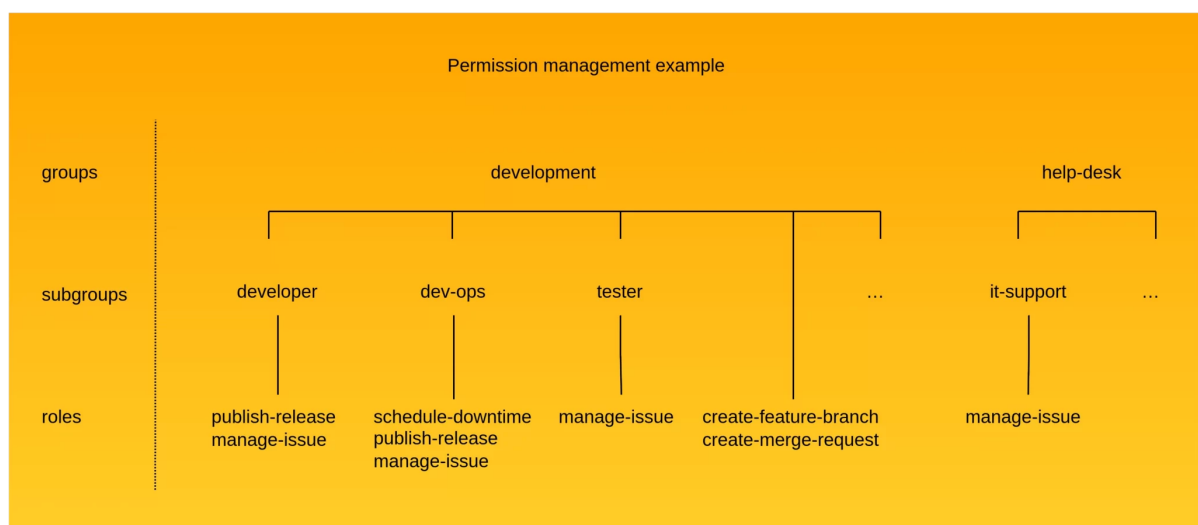
In order to simplify user management, we can utilize privilege inheritance. According to the hierarchy described in the documentation:

A group can have multiple subgroups but a group can have only one parent. Subgroups inherit the attributes and role mappings from their parent. Users inherit the attributes and role mappings from their parent as well.

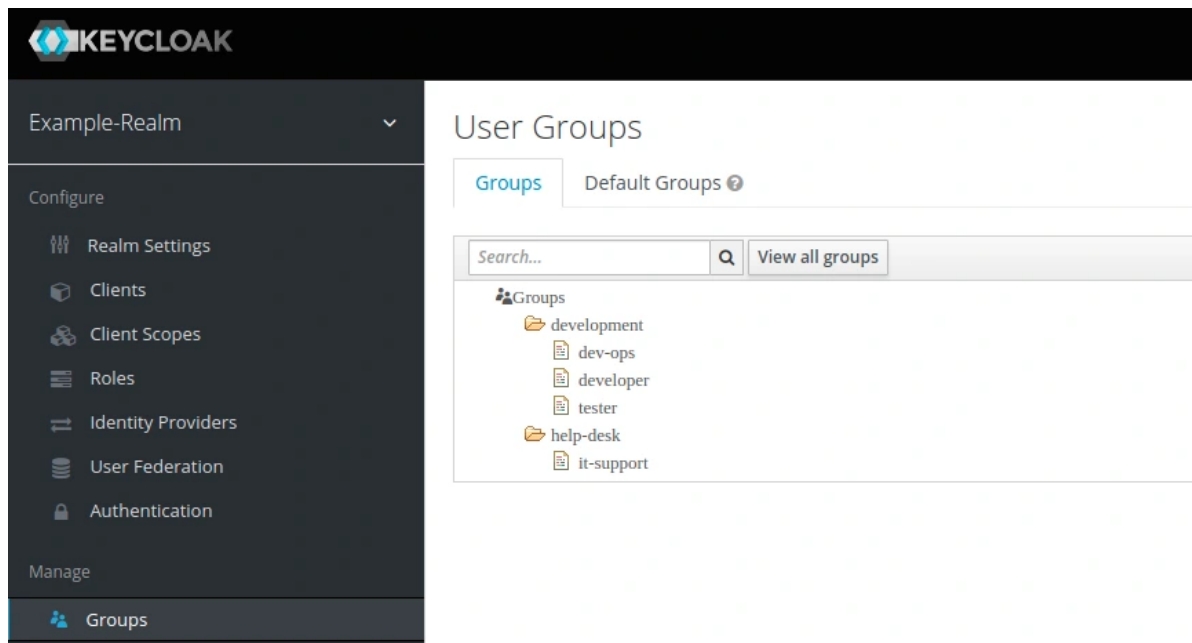
If you have a parent group and a child group, and a user that belongs only to the child group, the user in the child group inherits the attributes and role mappings of both the parent group and the child group.

https://www.keycloak.org/docs/16.0/server_admin/#proc-managing-groups_server_administration_guide

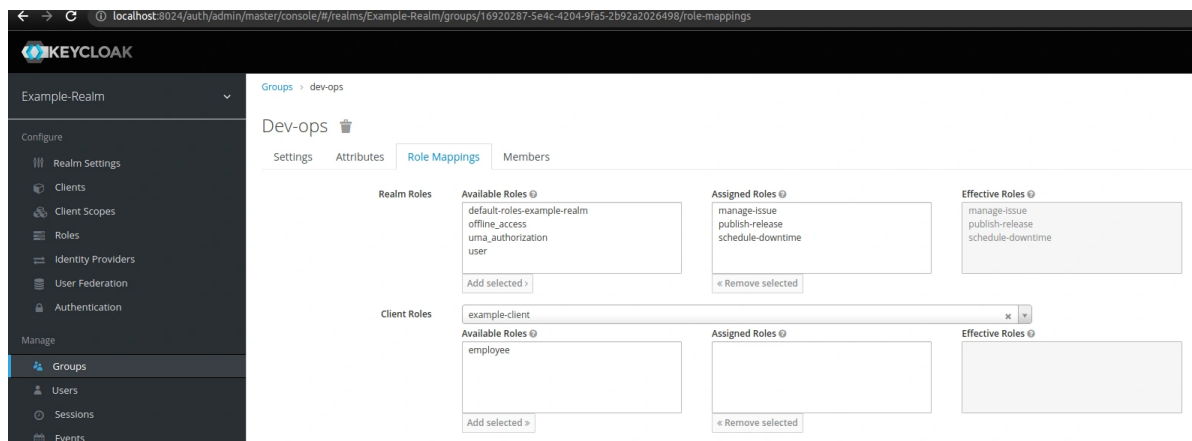
Let's explore this concept on a practical example. As an illustration, I want to assign privileges to users according to the following scheme:



First, I'm going to create all the roles (**publish-release**, **manage-issue**, etc.). For this example, I am creating realm-level roles.



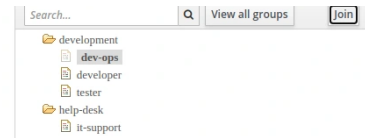
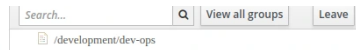
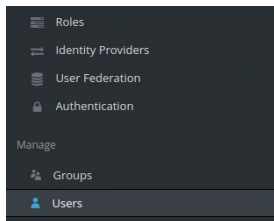
Then we can assign our roles to the created groups. To do this, edit a group, go to the **Role Mappings** tab, select the roles from the **Available Roles** list and click **Add selected**. In the screenshot below, you can see the role mapping for the **dev-ops** group:



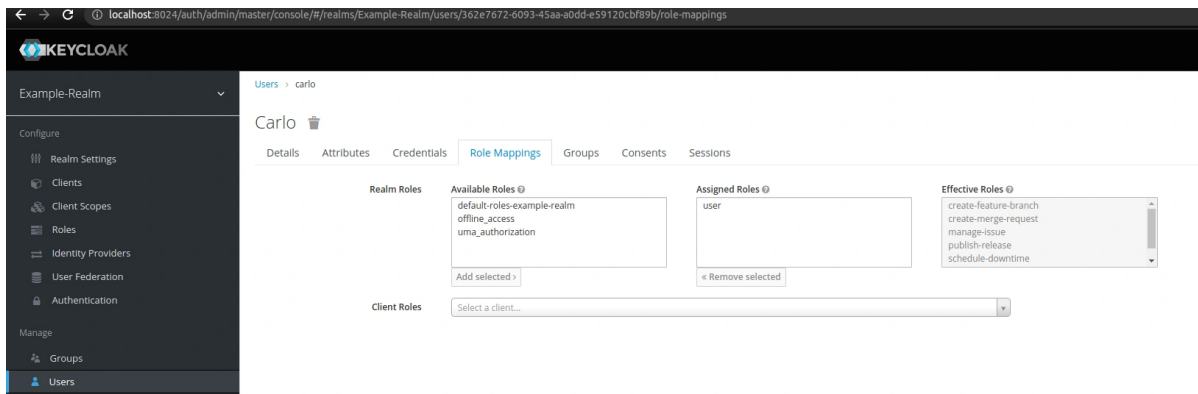
Finally, I'm going to assign one of the realm users, carlo, to the **dev-ops** group:



Q Search ...



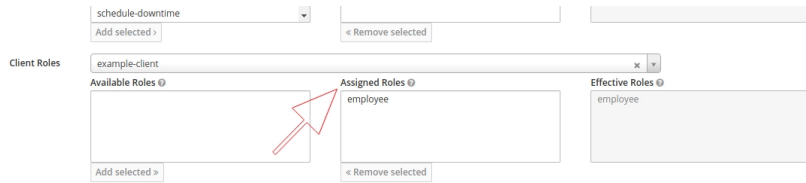
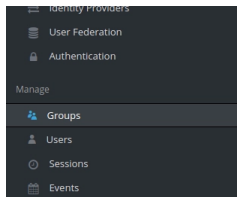
When we examine his role mappings, we see that his effective roles include not only those directly related to the **dev-ops** group (**manage-issue**, **publish-release**, **schedule-downtime**), but also those assigned to the **development** group (**create-feature-branch**, **create-merge-request**):



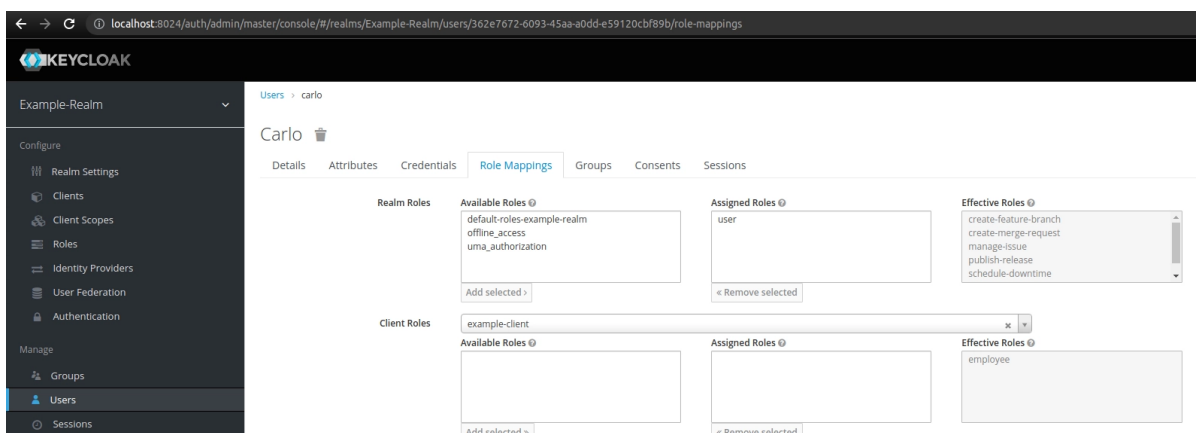
Moreover, the only role I have manually assigned to him is **user**. Therefore, this is the only role I can manually remove from his role mappings. To remove roles that he has acquired through membership in a specific group, we need to remove him from that group.

Combining realm and client level roles

Earlier in this article, I created a client-level role – the **employee** role. I can assign this role to all members of the development group:



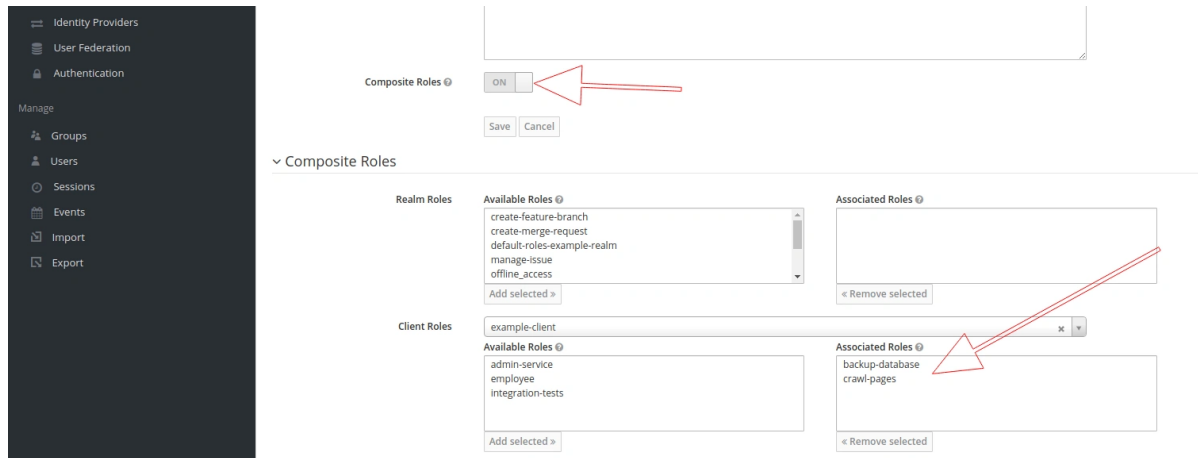
As a result, members will have realm-level roles recognized by all clients (e.g. **create-feature-branch**) and a client-level role (**employee**) visible only to the **example-client** application. We can see the result in the effective roles mappings for carlo:



Composite roles

Both realm and client-level roles can be marked as **Composite**. Thanks to this, we can assign them additional roles. In other words, a service with a composite role obtains all the roles associated with that composite. It's important to realise that we shouldn't overuse composites as recursive inheritance can lead to convoluted mappings.

Let's create a client-level role – **admin-service**, mark it as a composite role and associate it with the **backup-database** and **crawl-pages** roles:


 Search ...


Consequently, by assigning the **admin-service** role to an application, we also grant it all associated roles. As we can see, we can easily assign a complex set of permissions to services and applications.

Groups vs composite roles

The functions provided by the groups and the composite roles overlap and we technically can achieve the same goal with each of them. However, we should manage users with the former and applications with the latter. As we can read in the docs:

Composite Roles are similar to Groups as they provide the same functionality. The difference between them is conceptual. Composite roles apply the permission model to a set of services and applications. **Use composite roles to manage applications and services.**

Groups focus on collections of users and their roles in an organization. **Use groups to manage users.**

https://www.keycloak.org/docs/16.0/server_admin/#con-comparing-groups-roles_server_administration_guide

In addition, in the keycloak-dev mailing list archives from 2015 we can read:

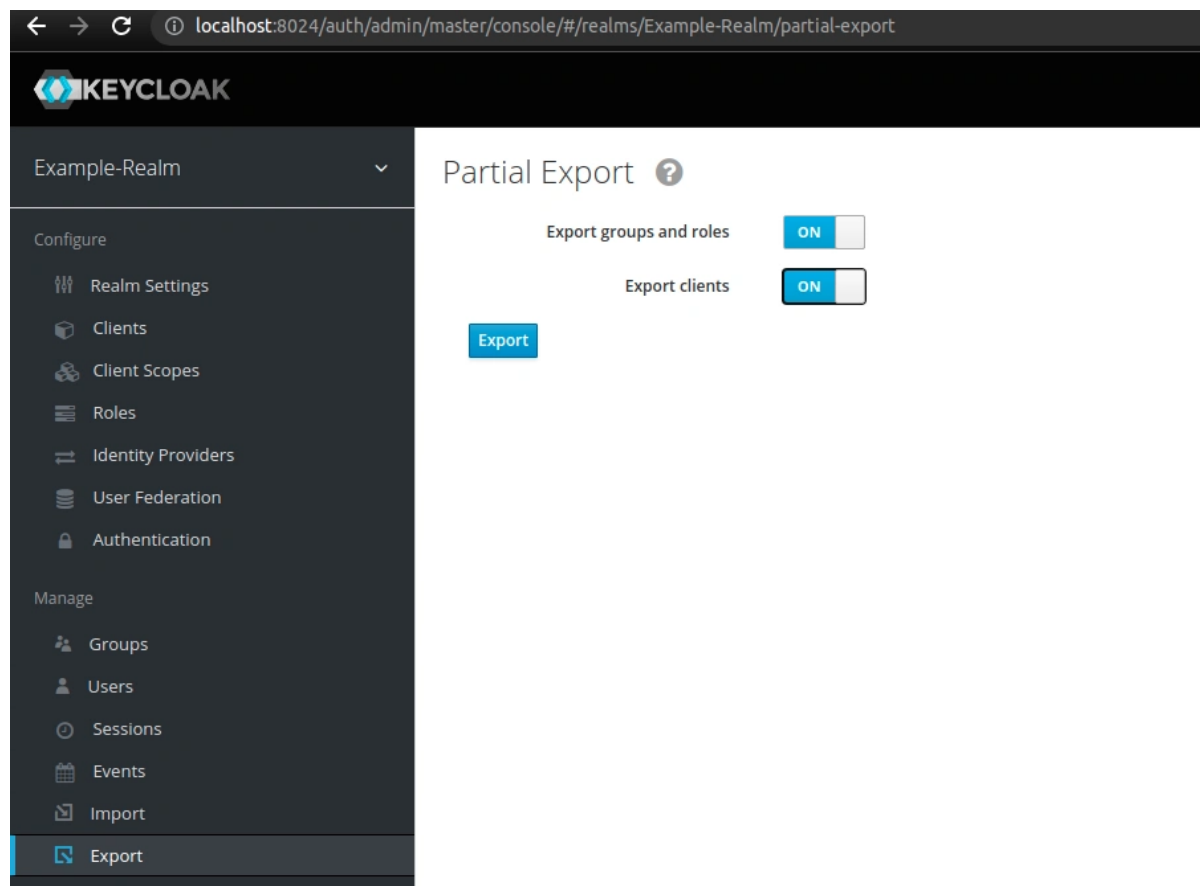
So, keycloak composite roles would be a way to organise rights for a set of applications. (...) Conversely, a keycloak group would provide a way to

<https://lists.jboss.org/pipermail/keycloak-dev/2015-November/005735.html>

Exporting Keycloak realm with user roles

Keycloak [Admin console](#) supports exporting realms. The following example shows how to save a realm with the groups, roles and its clients that we've created so far.

First, select the **Export** option from the menu and check the **Export groups and roles** and **Export clients** checkboxes:



You'll get the `realm-export.json` file that you can save on your machine. Inside you'll find the roles for `Example-Realm` and `example-client`:

1 | {



```
7      "id": "a498ea58-eb70-4c6f-a7f7-fefb0c3feaf3",
8      "name": "create-merge-request",
9      "description": "Can create a merge request in the repo.",
10     "composite": false,
11     "clientRole": false,
12     "containerId": "Example-Realm",
13     "attributes": {}
14 },
15 {
16     "id": "1d5a18c3-7780-4357-b41b-77d8b7083433",
17     "name": "offline_access",
18     "description": "${role_offline-access}",
19     "composite": false,
20     "clientRole": false,
21     "containerId": "Example-Realm",
22     "attributes": {}
23 },
24 {
25     "id": "40ac85fc-3804-49fc-870e-be141b60350e",
26     "name": "create-feature-branch",
27     "description": "Can create a feature branch in the repo.",
28     "composite": false,
29     "clientRole": false,
30     "containerId": "Example-Realm",
31     "attributes": {}
32 },
33 {
34     "id": "0e9b7666-2ad3-4393-a962-fa7c0652bc2b",
35     "name": "schedule-downtime",
36     "description": "Can manage the downtime schedule for the app.",
37     "composite": false,
38     "clientRole": false,
39     "containerId": "Example-Realm",
40     "attributes": {}
41 },
42 ...
```



```
48     "name": "employee",
49     "description": "Can view their own work schedule.",
50     "composite": false,
51     "clientRole": true,
52     "containerId": "97bfdbcb-6376-4b04-8a69-34fd8c6cd73b",
53     "attributes": {}
54 },
55 {
56     "id": "2530963f-f7a8-454f-895d-51b804a55248",
57     "name": "backup-database",
58     "description": "Run a scheduled automated database backup.",
59     "composite": false,
60     "clientRole": true,
61     "containerId": "97bfdbcb-6376-4b04-8a69-34fd8c6cd73b",
62     "attributes": {}
63 },
64 {
65     "id": "a37df70c-8dd9-4aa3-816f-759f830e669a",
66     "name": "admin-service",
67     "description": "Perform automated administration tasks.",
68     "composite": true,
69     "composites": {
70         "client": {
71             "example-client": [
72                 "backup-database",
73                 "crawl-pages"
74             ]
75         }
76     },
77     "clientRole": true,
78     "containerId": "97bfdbcb-6376-4b04-8a69-34fd8c6cd73b",
79     "attributes": {}
80 },
81 {
82     "id": "5aa70d72-d5ce-4feb-9f22-6c1e63ac987c",
83     "name": "crawl-pages",
```



```
89     }  
90   ],  
91   ...
```

If you don't want to lose these roles when you delete the **keycloak** containers and their associated volumes, replace the **realm-export.json** file assigned to the following volume in the **docker-compose-keycloak.yml** file:

```
1 # docker-compose-keycloak.yml  
2 services:  
3   keycloak:  
4     ...  
5     volumes:  
6       - ./keycloak/realms/realm-export.json:/tmp/realm-export.json
```

However, we won't be able to include users in the file created from the Admin console export. Therefore, if you need to export users as well, follow directions provided in the [Keycloak in Docker #5 – How to export a realm with users and secrets](#) post or [Importing and exporting the database](#) section of the documentation.

If you want to keep the default users and still export realm from the Admin console, just copy the **users** array from the **default-users.json** file to your new **realm-export.json** file.

Read more on user roles in Keycloak

- [Realm roles in Keycloak docs](#)
- [Client roles in Keycloak docs](#)
- [Groups in Keycloak docs](#)
- [Composite roles in Keycloak docs](#)
- [Keycloak with Spring Boot #4 – Simple guide for roles and authorities](#)
- [Role-Based Access Control to REST API with Keycloak](#)



 Search ...

One thought on "Keycloak in Docker #4 – How to define user privileges and roles"



Itamar says:
3rd January 2022 at 3:00 am

Nice Article

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

B**I**U

☰

☰
1
2
3

☰

☰

“

✂

📄

📄

🔗

✖

Name *

Email *

Website

☐ By using this form you agree with the storage and handling of your data by this website. *



[JavaTools](#)[Taking care](#)[Angular](#)

About [in](#) [on](#) [with](#)



Created by [Marta Szymek](#)

[Privacy Policy](#)