Keycloak for Node js

> **Product Name:** keycloak
> **Phase:** Authentication & Authorization
> **Date:** Nov 16, 2022
> **Creator** : Fayaz

Securing Node.js Express REST APIs with Keycloak

What is Keycloak?

- Keycloak is an open-source identity and access management solution which makes it easy to secure modern applications and services with little to no code.
- Keycloak comes with its own adapters for selected platforms, but it is also possible to use generic OpenID Connect Relying Party and SAML Service Provider libraries. But using the Keycloak Client Adaptors would be much simpler, and easy to use and they require less boilerplate code than what is typically required by a library.

Keycloak Configuration:

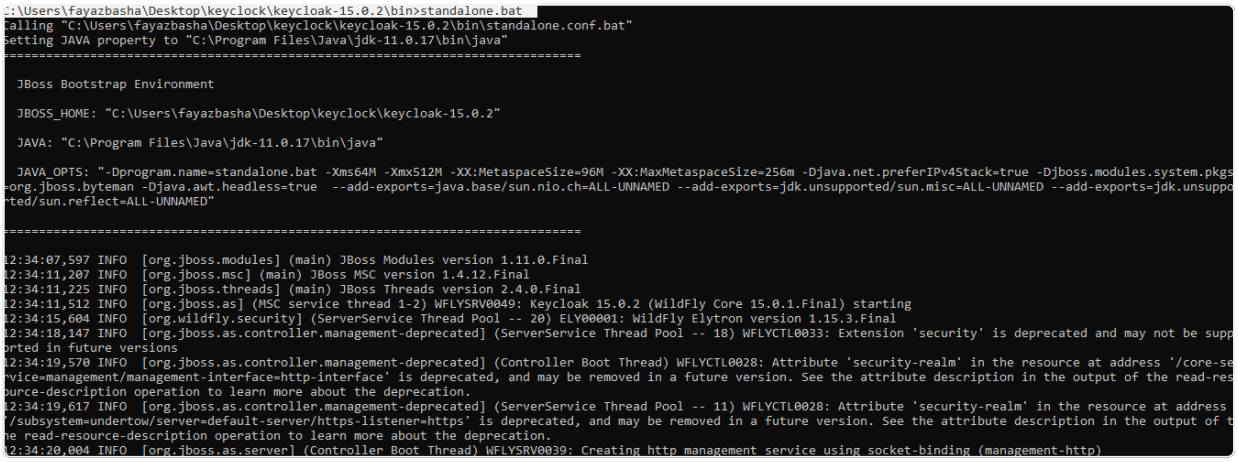
Setting Up a Keycloak Server

1. Download and install keycloak

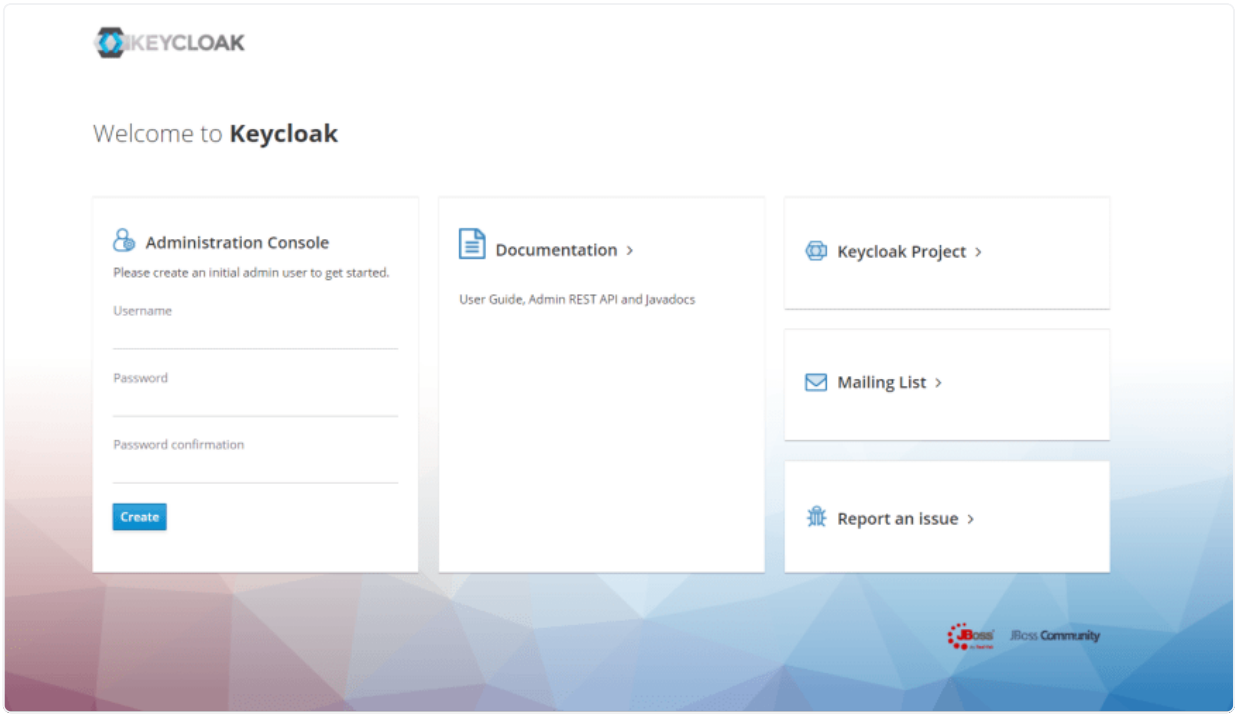
- Download the latest version of keycloak from here .
- Take the zip file
- At the time of writing this, I am using keycloak 15.0.2

If java is not installed,

- Install Java SDK
- Add the path to environmental variables
    - After Installing the zip file you should unzip it. If you are using the Linux server you should use this command to unzip `unzip keycloak-15.0.2.zip .`
    - After unzipping go to the bin folder and click on the standalone.bat file. you will get the below screen.



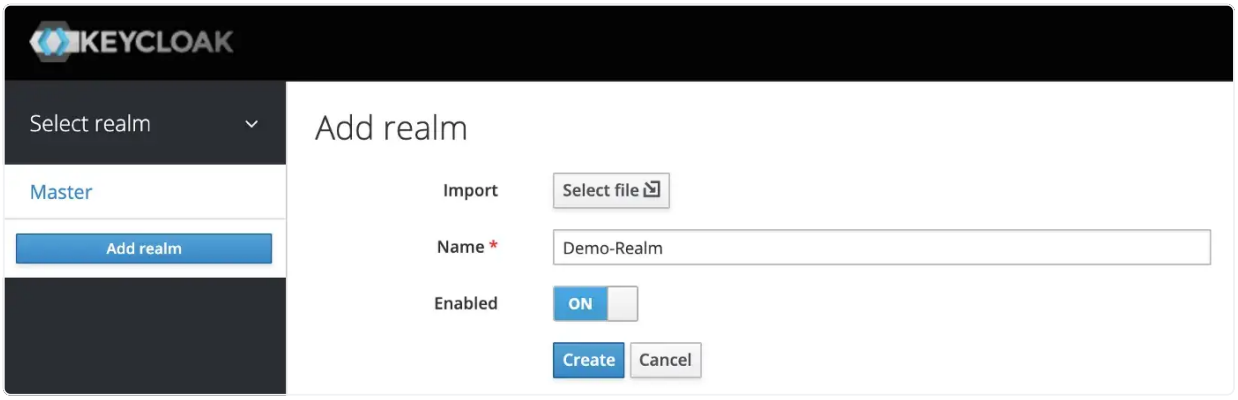
- Now let's open a browser and visit http://localhost:8080. We'll be redirected to http://localhost:8080/auth to create an administrative login as you can see below.



- We can now proceed to the Administrative Console. On the login page, we'll enter the initial admin user credentials

https://res.cloudinary.com/practicaldev/image/fetch/s--WWkvPPfM--/c_limit%...

1. From the **Master** drop-down menu, click **Add Realm**. When you are logged in to the master realm this drop-down menu lists all existing realms.
2. Type `Demo-Realm` in the **Name** field and click **Create**.

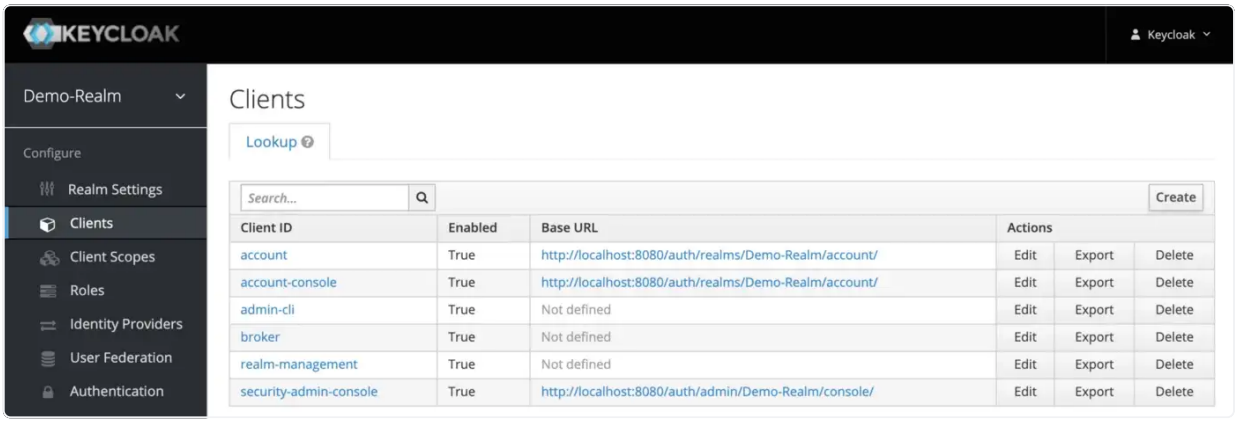


- When the realm is created, the main admin console page opens. Notice the current realm is now set to `Demo-Realm`. Switch between managing the `master` realm and the realm you just created by clicking entries in the **Select realm** drop-down menu.
- Make sure `Demo-Realm` is selected for the below configurations. Avoid using the master realm. You don't have to create the realm every time. It's a one time process.

Create a Client

Clients are entities that can request Keycloak to authenticate a user. Most often, clients are applications and services that want to use Keycloak to secure themselves and provide a single sign-on solution. Clients can also be entities that just want to request identity information or an access token so that they can securely invoke other services on the network that are secured by Keycloak.

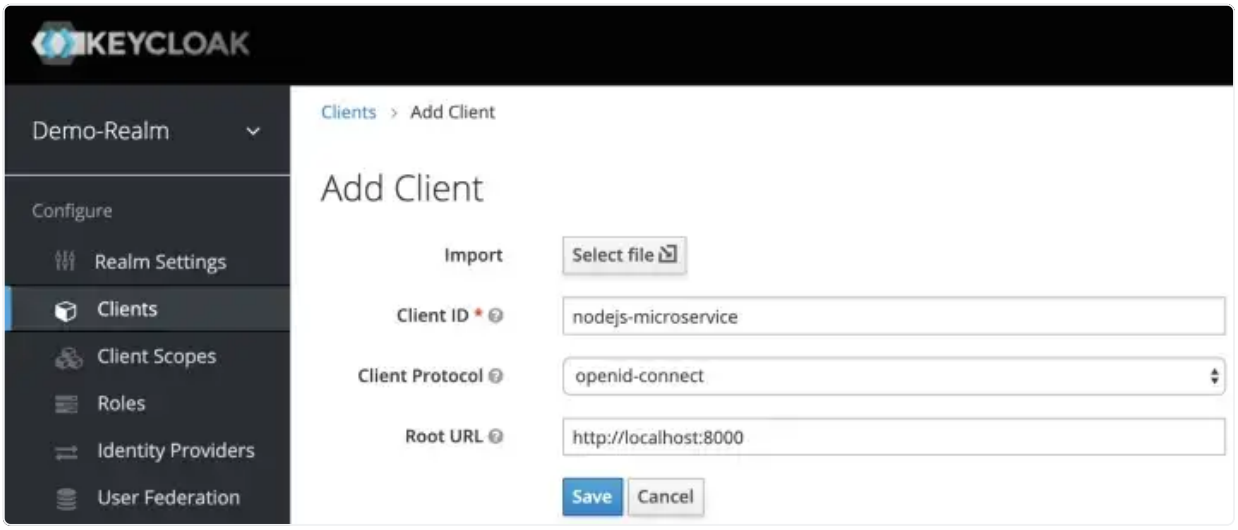
1. Click on the **Clients** menu from the left pane. All the available clients for the selected Realm will get listed here.



Client Management in Keycloak Admin Console

2. To create a new client, click **Create**. You will be prompted for a **Client ID**, a **Client Protocol** and a **Root URL**. A good choice for the client ID is the name of your application

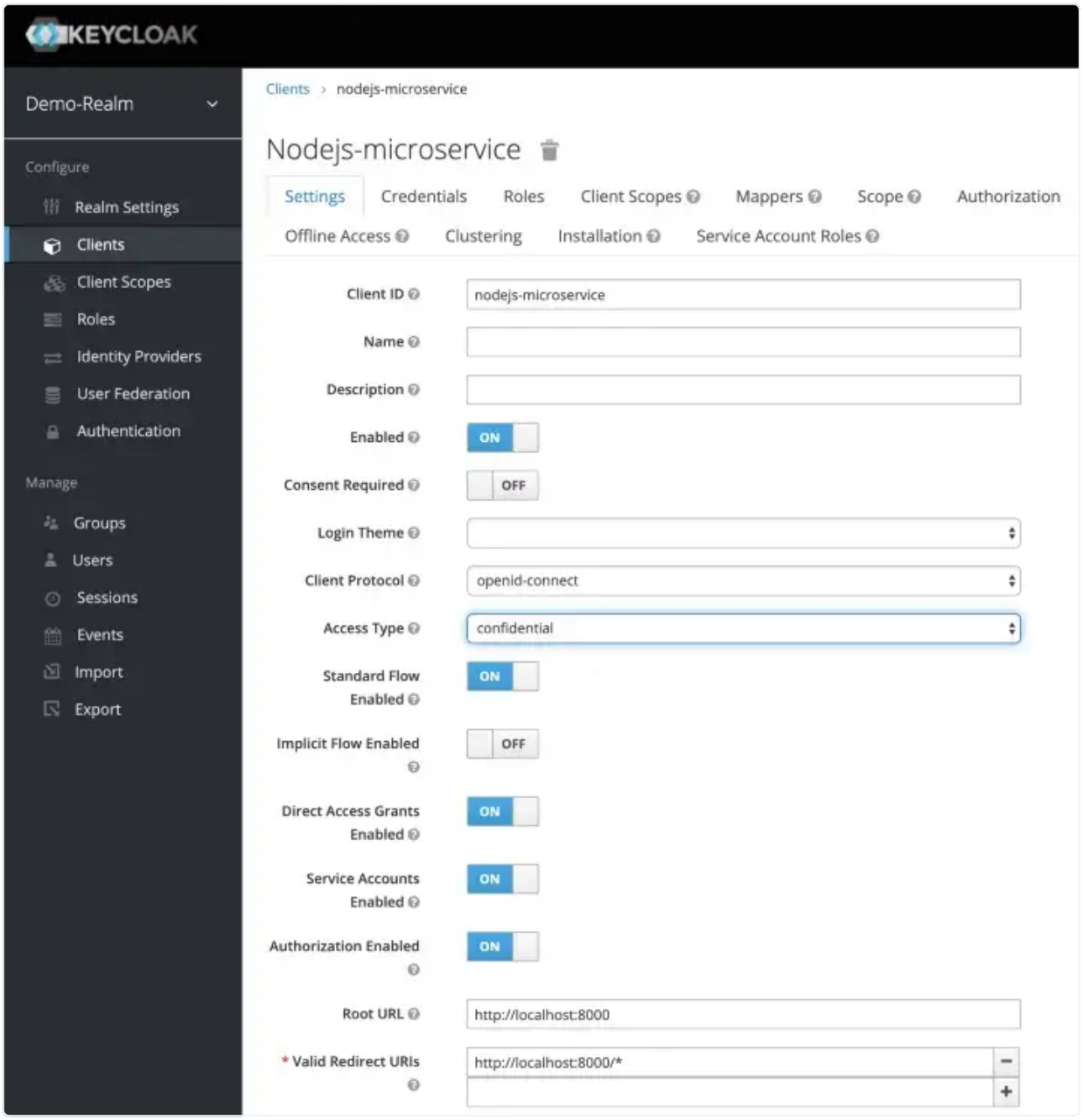
(`nodejs-microservice`), the client protocol should be set to `openid-connect` and the root URL should be set to the application URL.



Add Client in Keycloak Admin Console

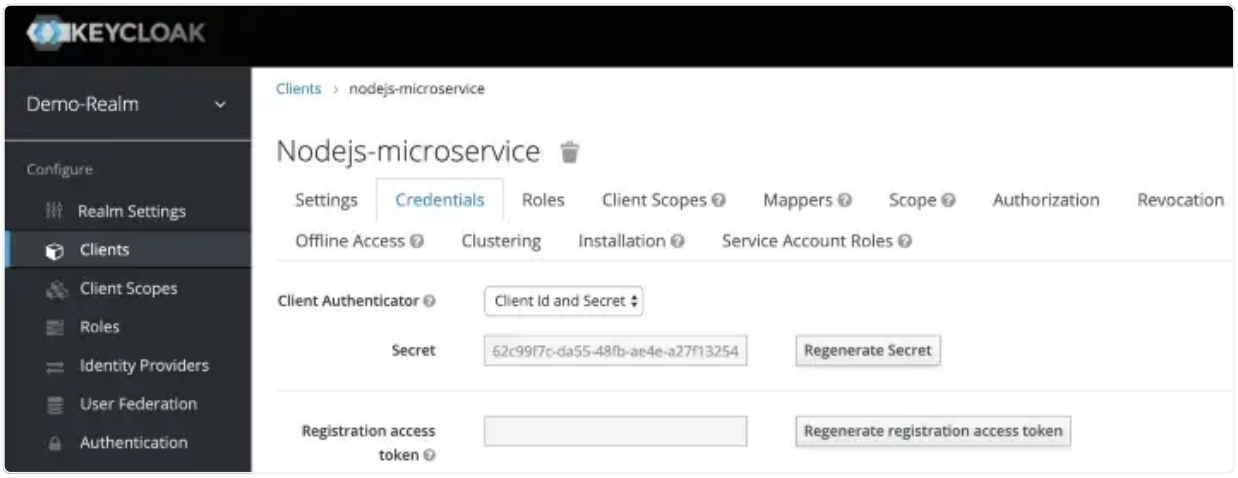
3. After saving you will be presented with the client configuration page where you can assign a name and description to the client if desired.

Set the **Access Type** to `confidential`, **Authorization Enabled** to `ON`, **Service Account Enabled** to `ON` and click **Save**.



Configure client with Access Type: 'confidential'

Credentials tab will show the **Client Secret** which is required for the Node.js Application Keycloak configurations.

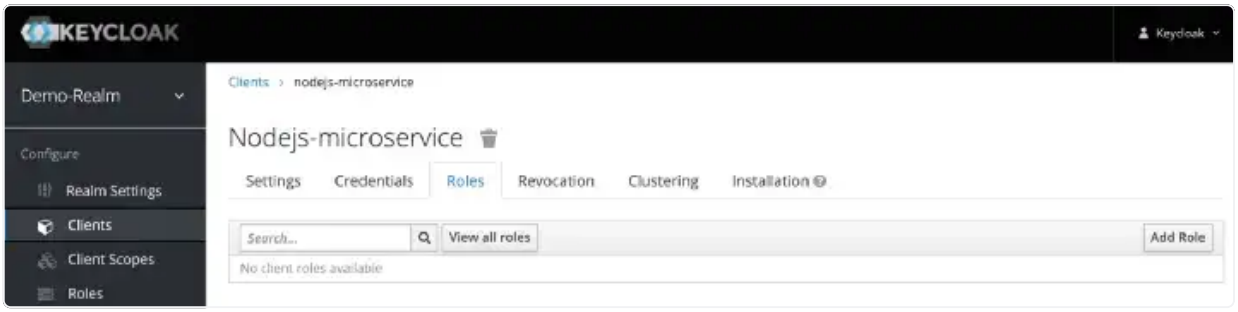


Client Credentials Tab

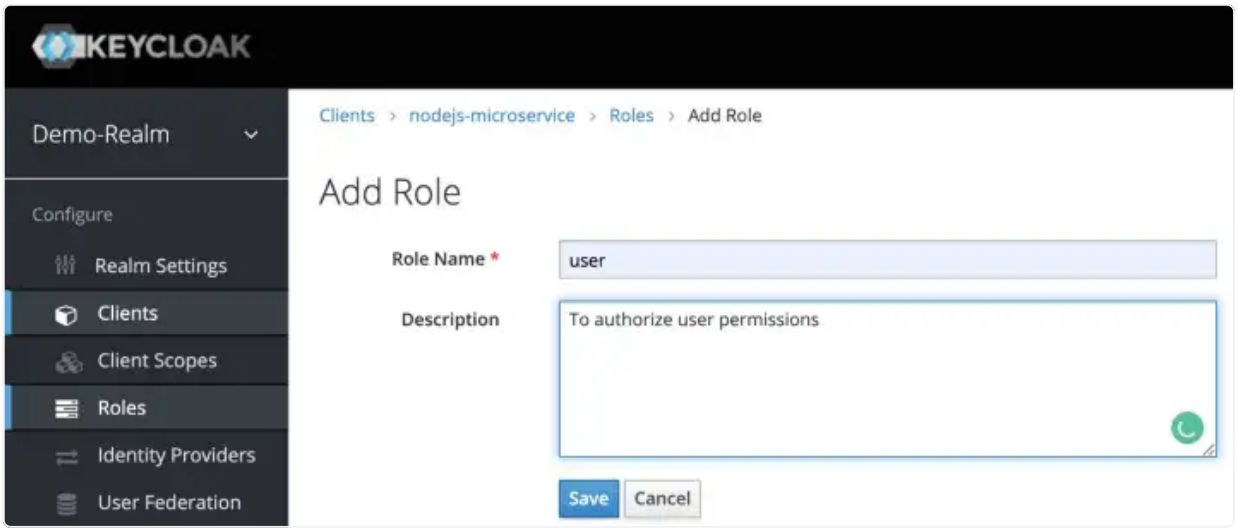
4. Go to **Client Roles** tab to create the `nodejs-microservice` role definitions. Imagine the Application that you are building with have different types of users with different user permissions. Ex: users and administrators.

- Some APIs would only be accessible to users only.
- Some APIs would be accessible to administrators only.
- Some APIs would be accessible to both users and administrators.

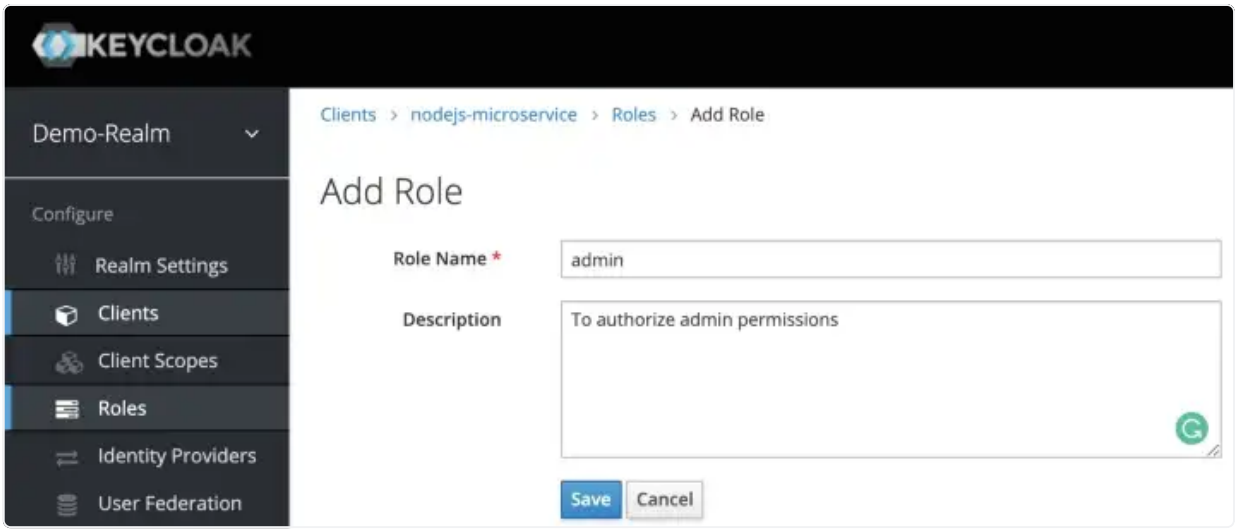
As per the example, let's create two roles: `user` and `admin` by clicking **Add Role** button.



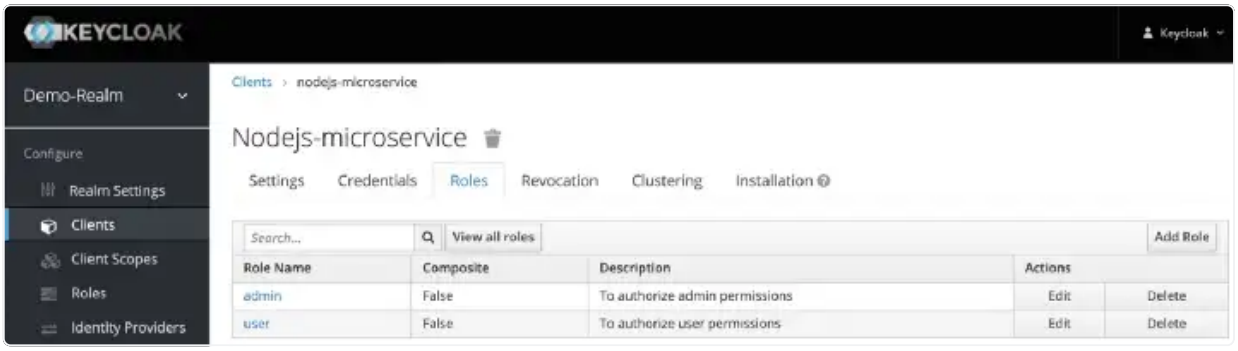
'nodejs-microservice' Client Roles



Add ‘user’ role and Save



Add ‘admin’ role and Save



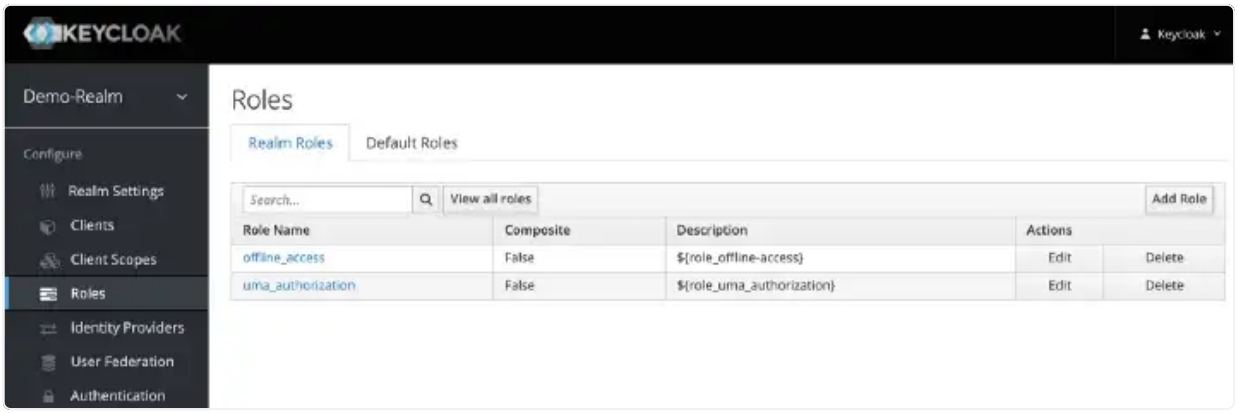
‘nodejs-microservice’ Client Roles after adding ‘user’, ‘admin’ roles

Create Realm Roles

Applications often assign access and permissions to specific roles rather than individual users as dealing with users can be too fine grained and hard to manage.

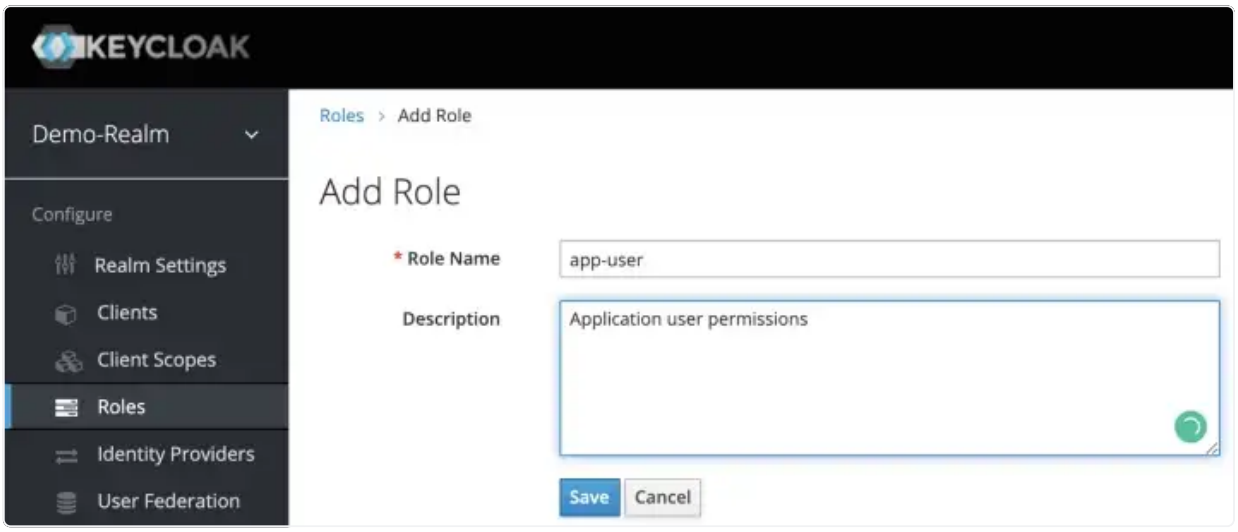
Let's create `app-user` and `app-admin` Realm roles by assigning corresponding `nodejs-microservice` roles (`user`, `admin`).

1. Click on the **Roles** menu from the left pane. All the available roles for the selected Realm will get listed here.



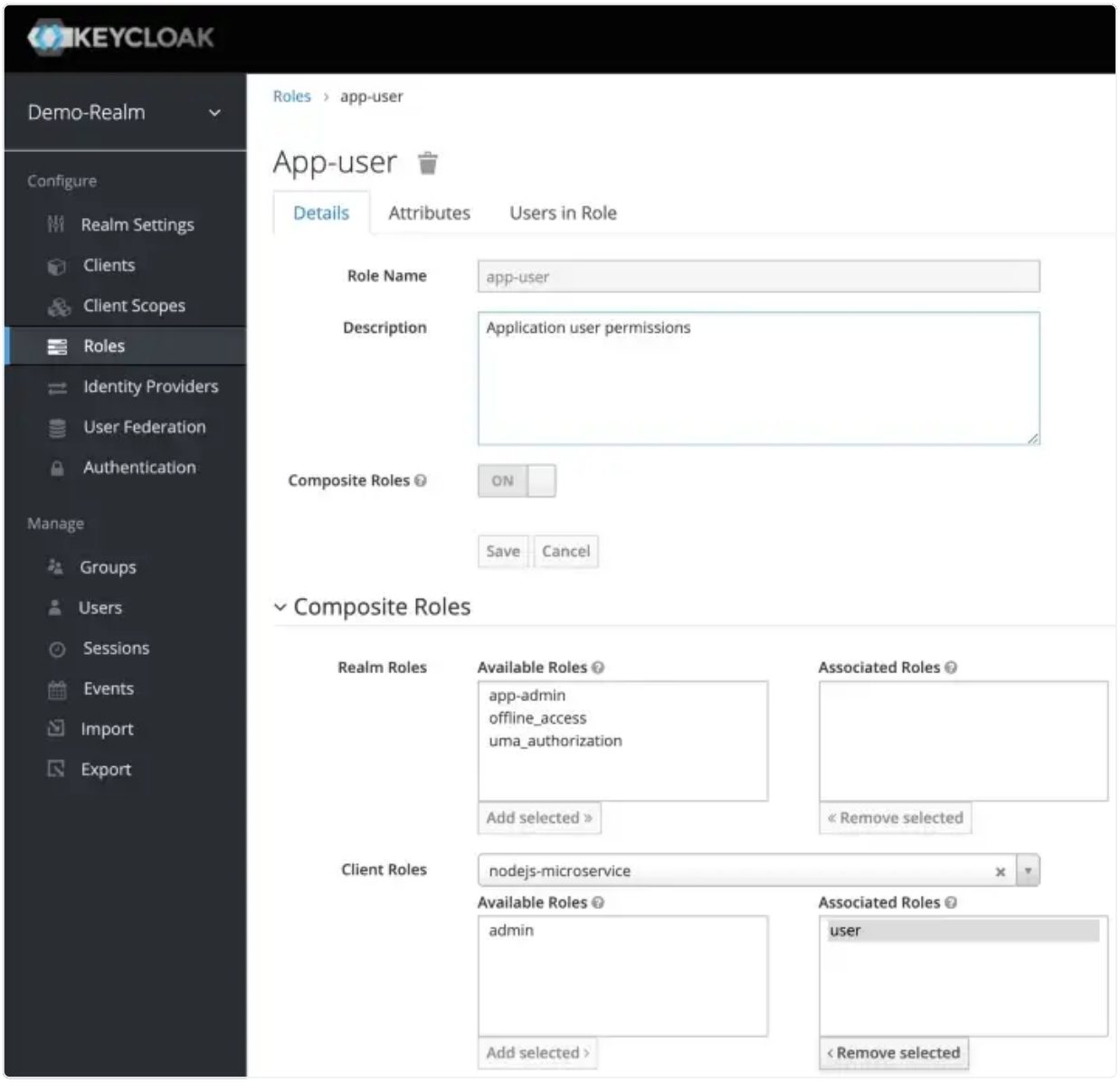
Realm Roles in Keycloak Admin Console

2. To create `app-user` realm role, click **Add Role**. You will be prompted for a **Role Name**, and a **Description**. Provide the details as below and **Save**.



Adding 'app-user' Realm Role

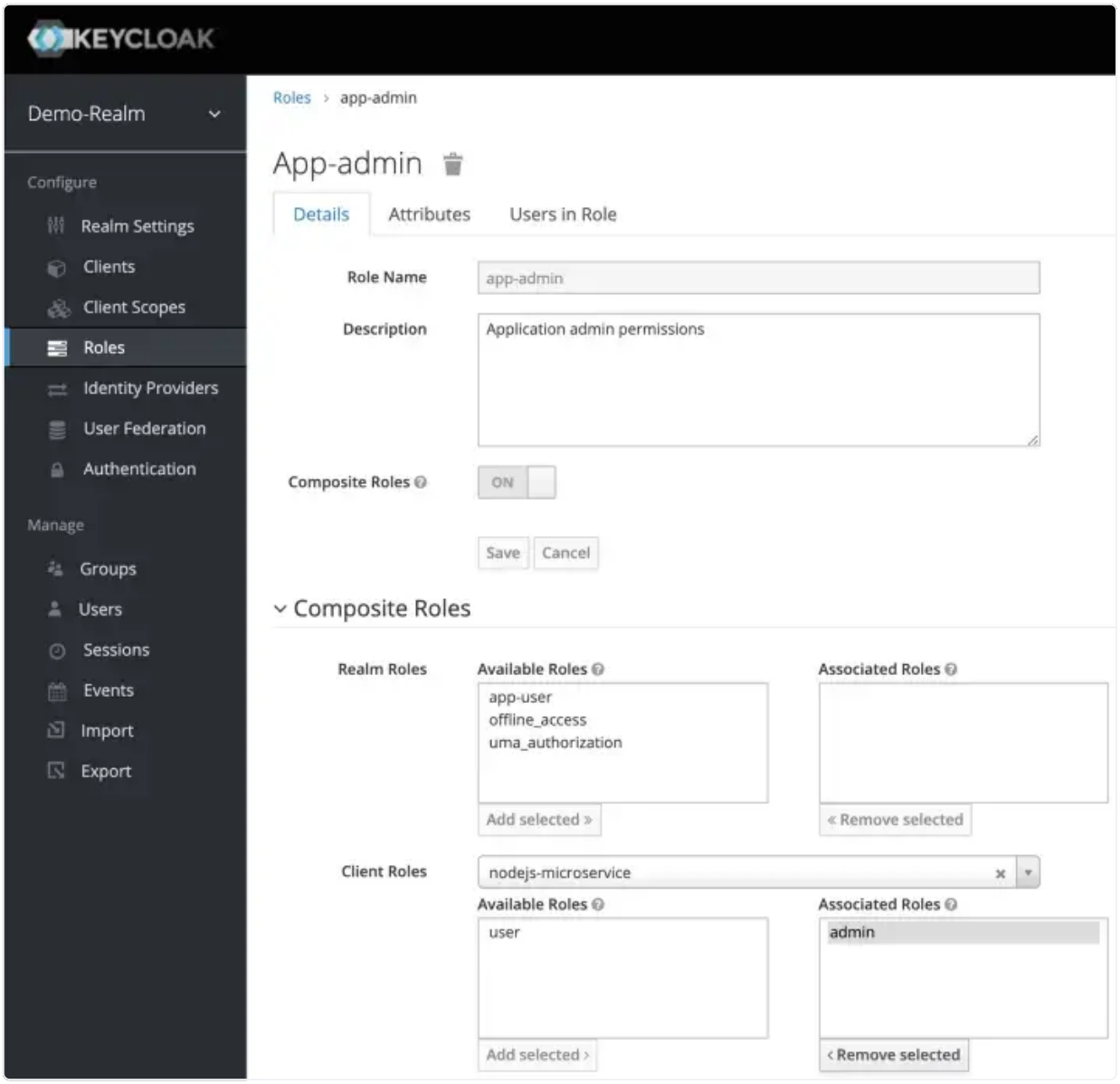
After **Save**, enabled **Composite Roles** and Search for `nodejs-microservice` under **Client Roles** field. Select `user` role of the `nodejs-microservice` and Click **Add Selected >**.



Assign ‘user’ Client Role to ‘app-user’ Realm Role

This configuration will assign `nodejs-microservice` `user` client role to the `app-user` realm role. If you have multiple clients with multiple roles, pick and choose the required roles from each client to create realm roles based on the need.

3. Follow the same steps to create the `app-admin` user but assign `admin` client role instead of `user` role.



Assign ‘admin’ Client Role to ‘app-admin’ Realm Role

Create Users

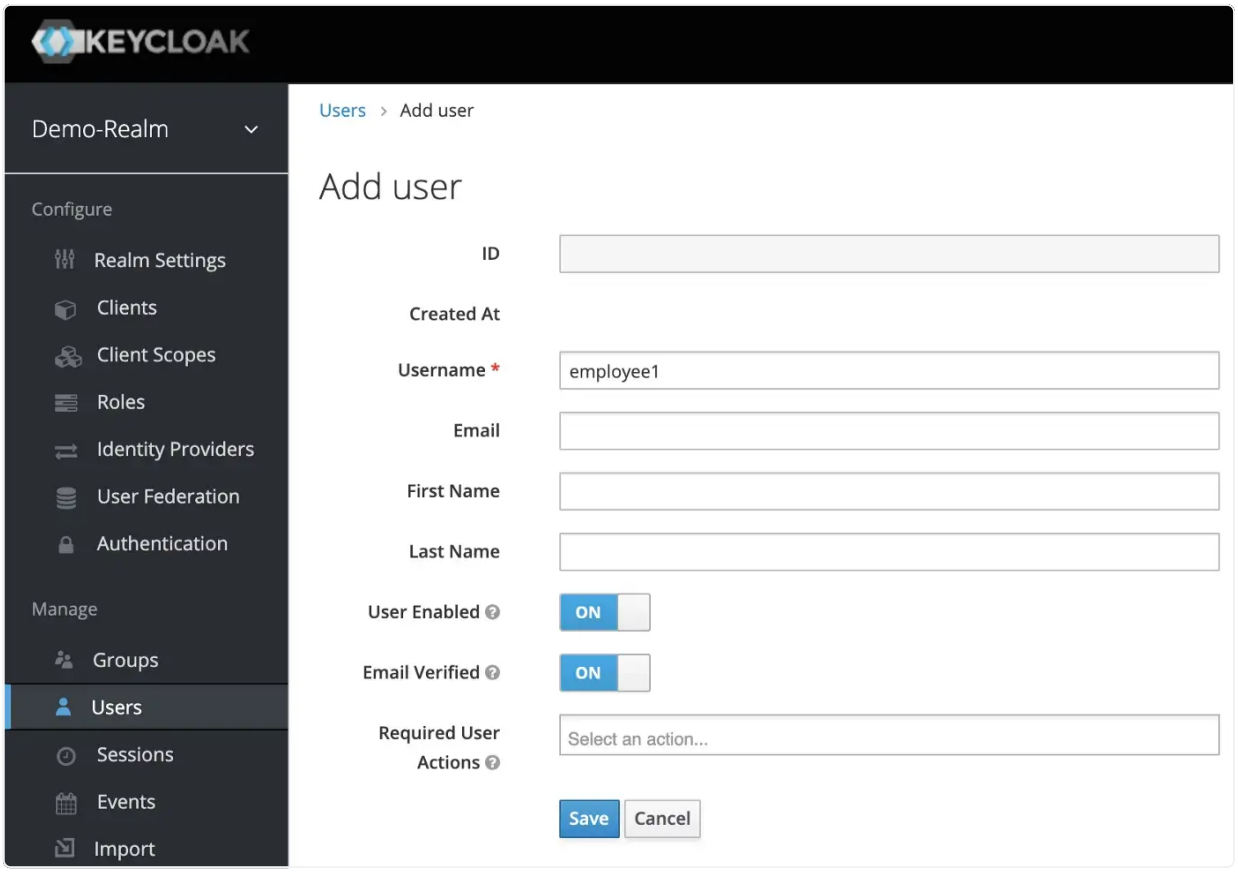
Users are entities that are able to log into your system. They can have attributes associated with themselves like email, username, address, phone number, and birth day. They can be assigned group membership and have specific roles assigned to them.

Let’s create following users and grant them `app-user` and `app-admin` roles for testing purposes.

- employee1 with `app-user` realm role

- employee2 with `app-admin` realm role
- employee3 with `app-user` & `app-admin` realm roles

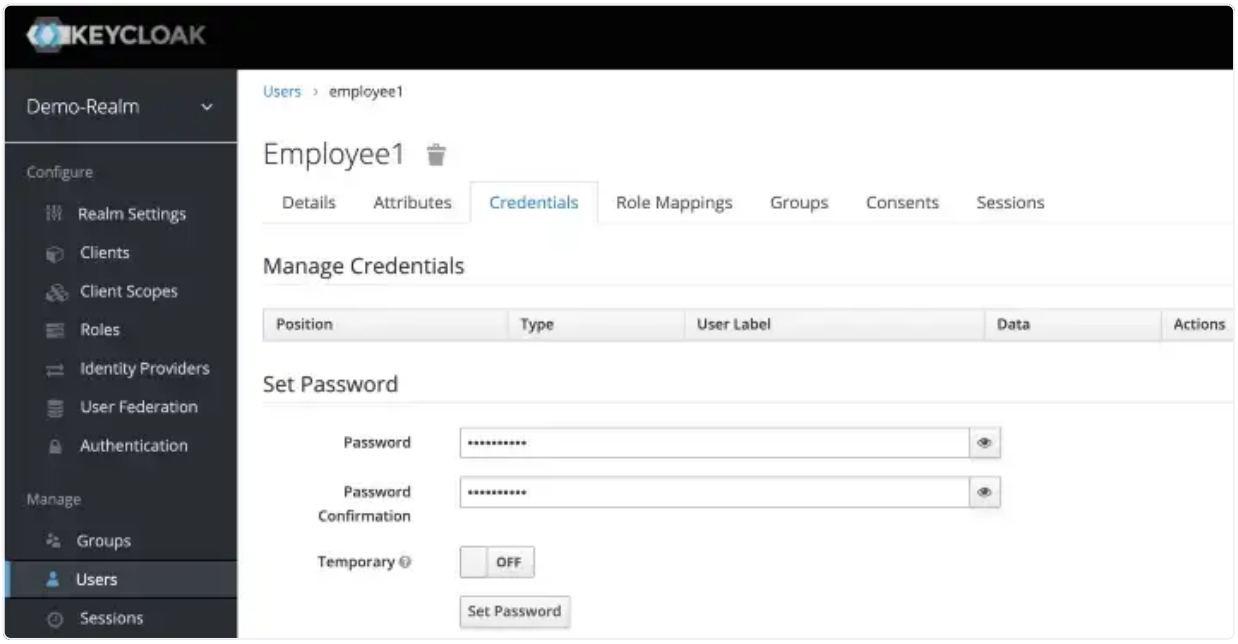
1. From the menu, click **Users** to open the user list page.
2. On the right side of the empty user list, click **Add User** to open the add user page.
3. Enter a name in the `Username` field; this is the only required field. Flip the **Email Verified** switch from **Off** to **On** and click **Save** to save the data and open the management page for the new user.



Add New User to 'Demo-Realm'

4. Click the **Credentials** tab to set a temporary password for the new user.

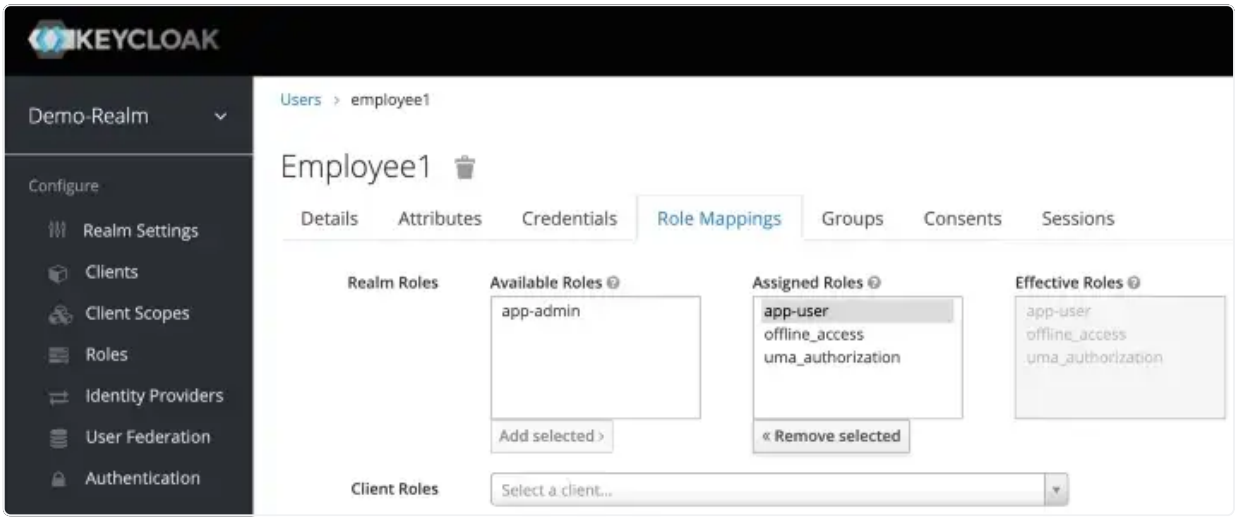
5. Type a new password and confirm it. Flip the **Temporary** switch from **On to** Off and click **Reset Password** to set the user password to the new one you specified. For simplicity let's set the password to `mypassword` for all the users.



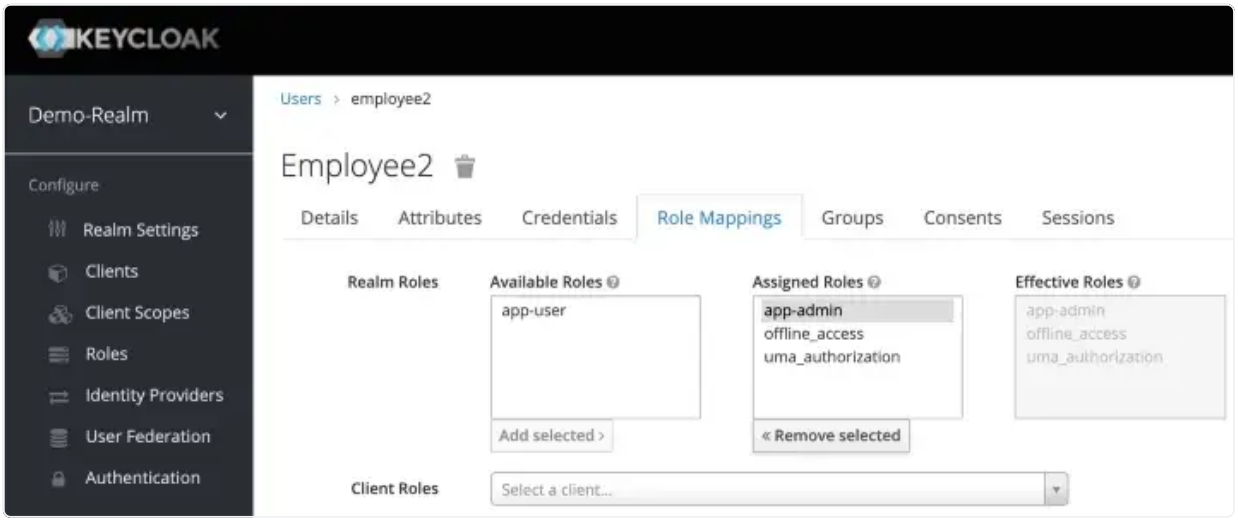
Setting Credentials to Users

6. Click the **Role Mappings** tab to assign realm roles to the user. Realm roles list will be available in **Available Roles** list. Select one required role and click on the **Add Selected >** to assign it to the user.

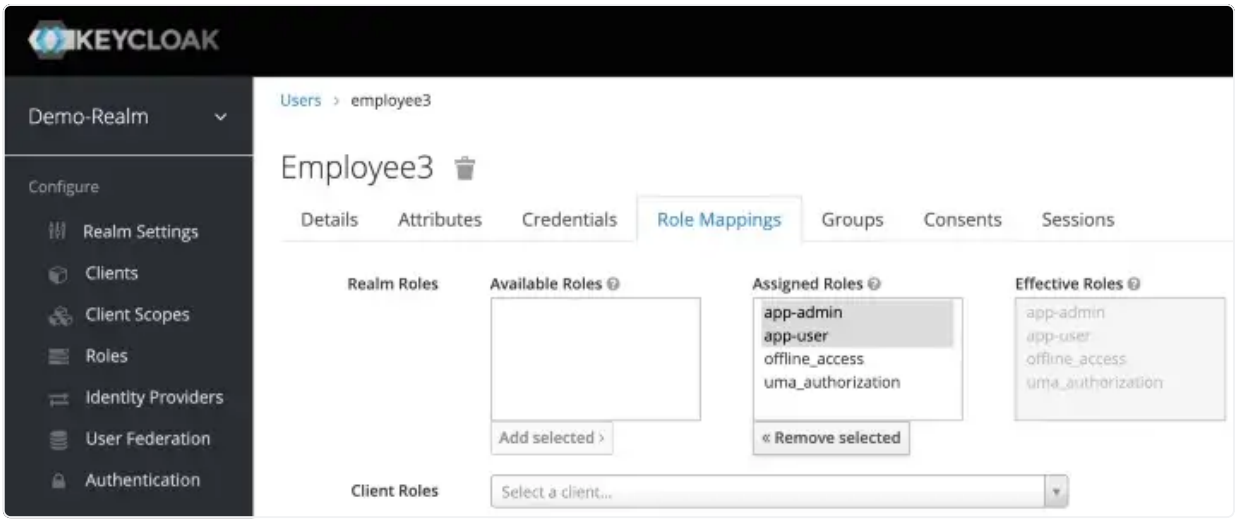
After role assignment, assigned roles will be available under **Assigned Roles** list. Role assignments for `employee1`, `employee2`, and `employee3` would be as below.



`employee1` Role Assignment



`employee2` Role Assignment



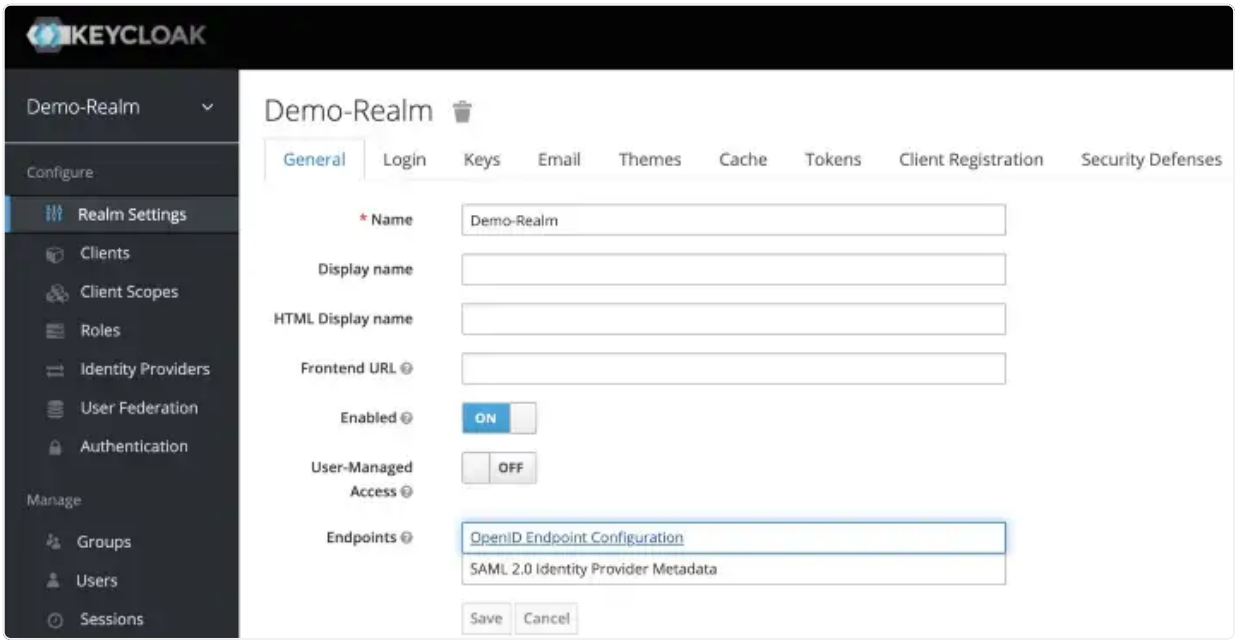
`employee3` Role Assignment

Yes, it was a bit of a hassle to go through all the configurations. But when you keep using Keycloak, these configurations will become a piece of cake. For new microservices getting added, you don't need to do all of the above. You just need to add a new client with client roles and assign the client roles to corresponding realm roles.

1.0:Generate Tokens

Let's learn how to generate an access token for Keycloak users.

1. Go to **Realm Settings** of the `Demo-Realm` from the left menu and click on **OpenID Endpoint Configuration** to view OpenID Endpoint details.



Realm Settings of 'Demo-Realm'


```
{
  "issuer": "http://localhost:8080/auth/realms/Demo-Realm",
  "authorization_endpoint": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/auth",
  "token_endpoint": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/token",
  "token_introspection_endpoint": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/token/introspect",
  "userinfo_endpoint": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/userinfo",
  "end_session_endpoint": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/logout",
  "jwks_uri": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/certs",
  "check_session_iframe": "http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/login-status-iframe.html",
  "grant_types_supported": [
    "authorization_code",
    "implicit",
    "refresh_token",
    "password",
    "client_credentials"
  ],
  "response_types_supported": [
    "code",
    "none",
    "id_token",
    "token",
    "id_token token",
    "code id_token",
    "code token",
    "code id_token token"
  ],
  "subject_types_supported": [
    "public",
    "pairwise"
  ],
  "id_token_signing_alg_values_supported": [
    "PS384",
    "ES384",
    "RS384",
    "HS256",
    "HS512",
    "ES256",
    "RS256",
    "HS384",
    "ES512",
    "PS256",
    "PS512",
    "RS512"
  ],
```


Keycloak Realm OpenID Endpoint Configuration

2. Copy **token_endpoint** from the **OpenID Endpoint Configuration**. URL would look like:

```
<KEYCLOAK_SERVER_URL>/auth/realms/<REALM_NAME>/protocol/openid-connect/tokenEx:
 http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/token
```

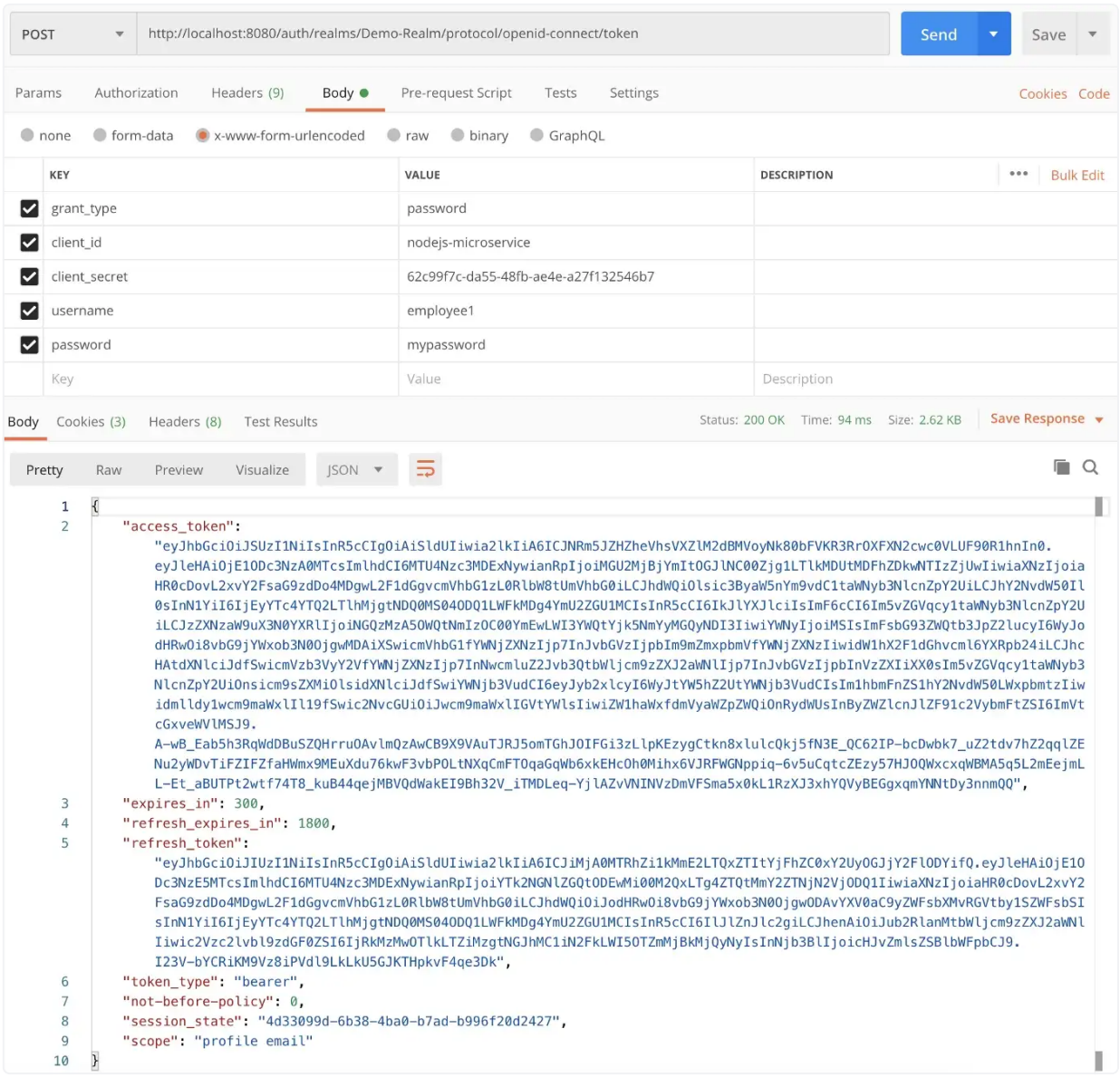
3. Use the following CURL command to generate user credentials.
Replace `KEYCLOAK_SERVER_URL`, `REALM_NAME`, `CLIENT_ID`, `USERNAME`, `PASSWORD` with correct values.

```
curl -X POST '<KEYCLOAK_SERVER_URL>/auth/realms/<REALM_NAME>/protocol/openid-connect/token' \
 --header 'Content-Type: application/x-www-form-urlencoded' \
 --data-urlencode 'grant_type=password' \
 --data-urlencode 'client_id=<CLIENT_ID>' \
 --data-urlencode 'username=<USERNAME>' \
 --data-urlencode 'password=<PASSWORD>'
```

Example:

```
curl -X POST 'http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/token' \
 --header 'Content-Type: application/x-www-form-urlencoded' \
 --data-urlencode 'grant_type=password' \
 --data-urlencode 'client_id=nodejs-microservice' \
 --data-urlencode 'client_secret=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxx' \
 --data-urlencode 'username=employee1' \
 --data-urlencode 'password=mypassword'
```

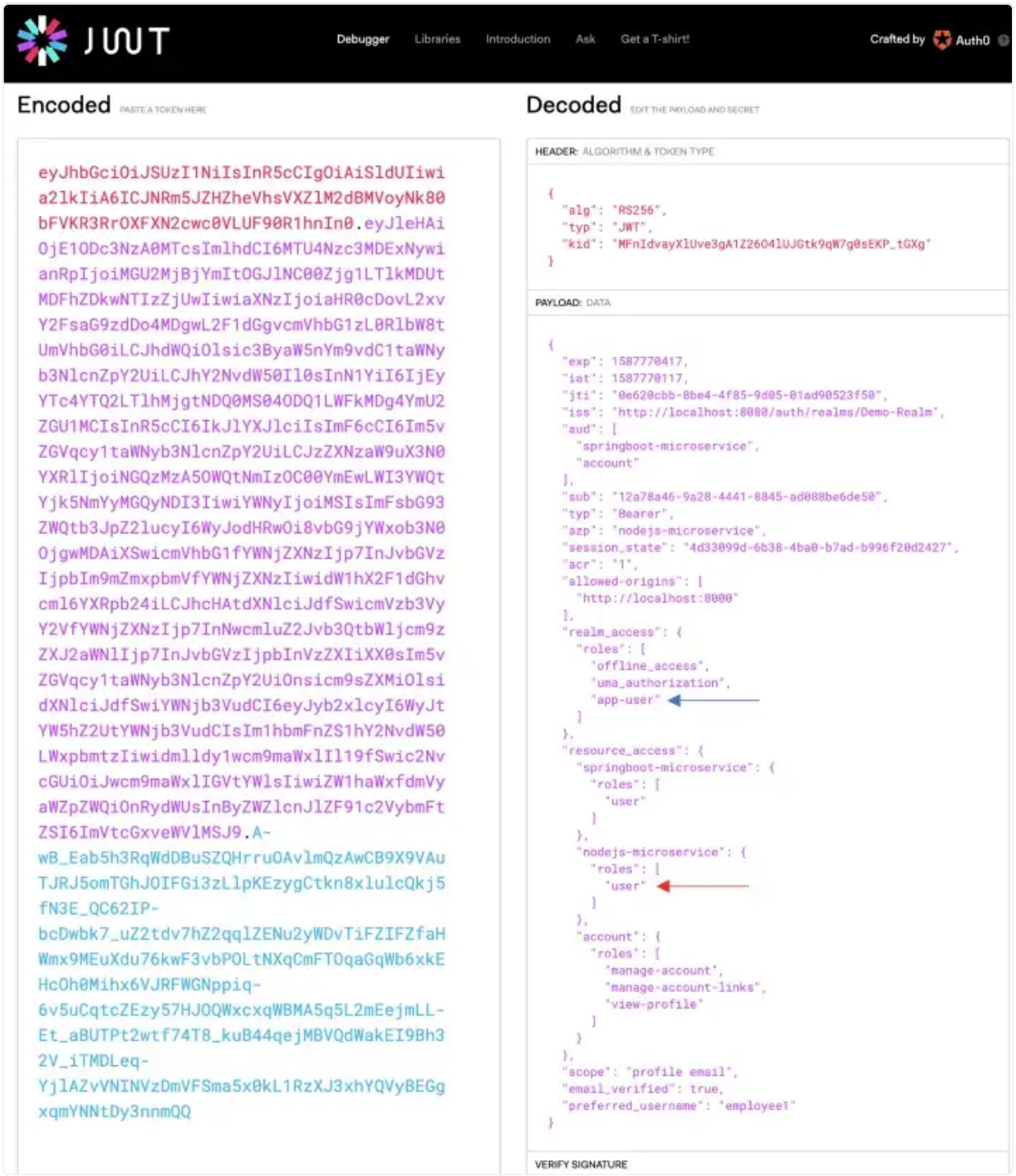
Execute the CURL from Terminal or use Postman. The response would look like below.



Get Token using Postman

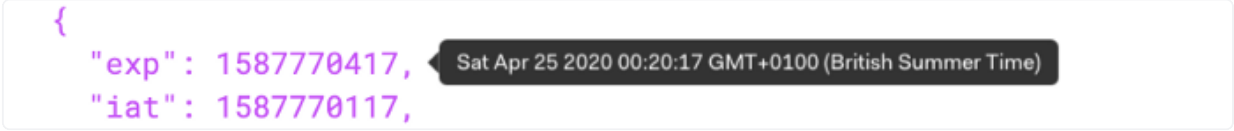
Let's decode the **access_token** JWT token issued for `employee1` using https://jwt.io. **access_token** includes the permission details.

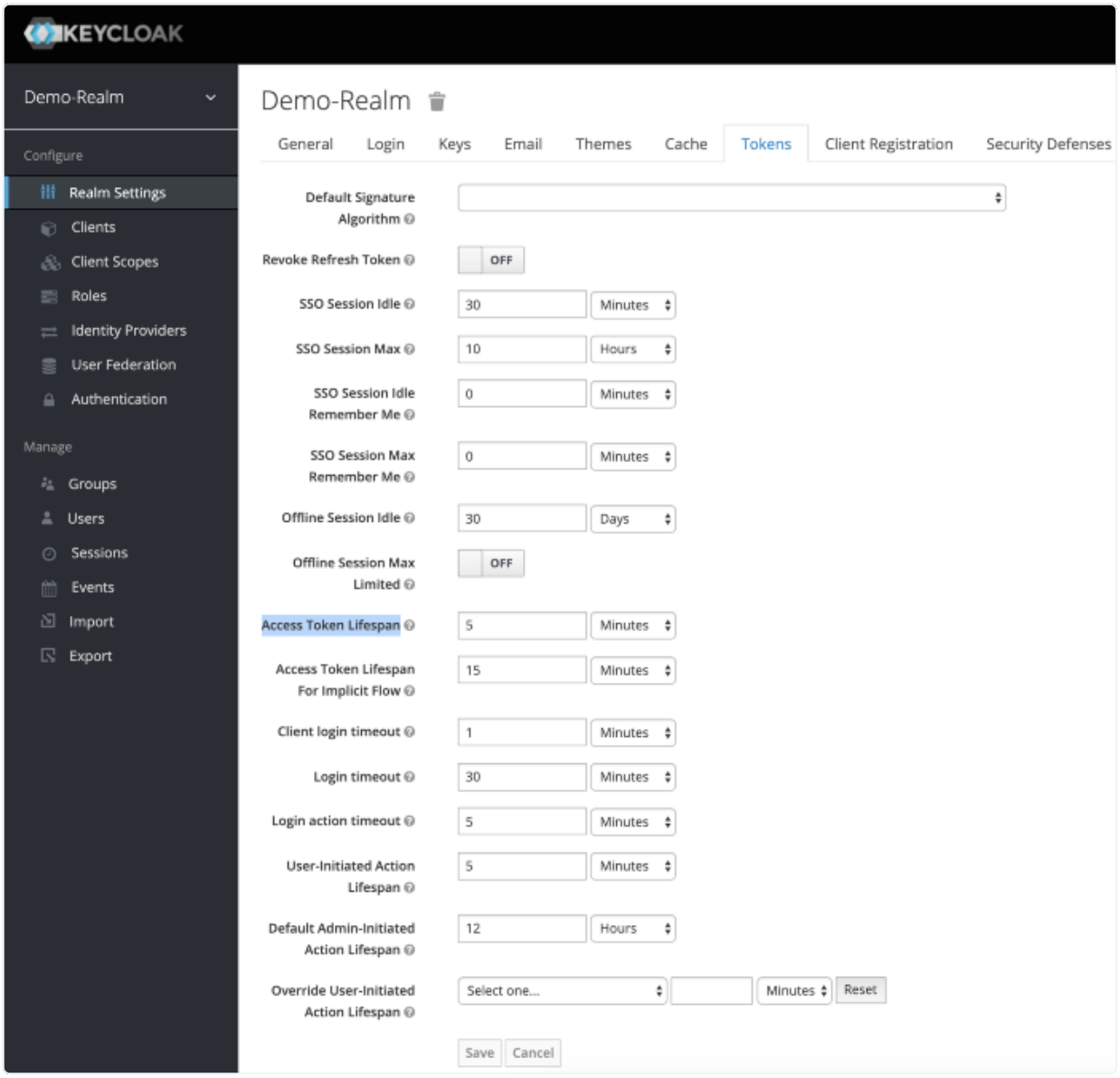
- **realm_access.roles** includes `app_user` realm role.
- **resource_access.nodejs-microservice.roles** include the `user` client role.
- **preferred_username** includes the username of the user (`employee1`)
-



Decoded access token

- **iat, exp** includes the token issued time as well as the token expiry time. Access Token expiry times can be customizable under **Realm Settings**, **Tokens** tab. By default, **Access Token Lifespan** would be set to 5 minutes which can be customized based on your security requirements.





In the testing phase of the Node.js Application, use the above steps to generate access tokens for multiple users with corresponding user credentials. Further, if the token expired, generate a new token with the same process.

Node.js Application Configuration

Let's build a new Node.js application and configure it with Keycloak Node.js Adaptor.

Creating the Node.js Application

Make sure Node.js is installed in your development environment.

- Create a node express project and Create a new file called `index.js` and with the below content.

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   res.send("Server is up!");
});

app.listen(4000);
```

This will start the server. To test this app, open your browser and go to **http://localhost:4000** and `Server is up!` message will appear in the browser.

Creating Test Controller

1. Create a new folder **`controller`** and create a new file **`test-controller.js`** in the created folder with the below content.

```
var express = require('express');
var router = express.Router();

router.get('/anonymous', function(req, res){
    res.send("Hello Anonymous");
});
router.get('/user', function(req, res){
    res.send("Hello User");
});

router.get('/admin', function(req, res){
    res.send("Hello Admin");
});

router.get('/all-user', function(req, res){
    res.send("Hello All User");
});

module.exports = router;
```

- Import `test-controller.js` and add the testController router to express in **index.js** before the **app.listen** function call.

```
var testController = require('./controller/test-controller.js');
```

```
app.use('/test', testController);
```

- The ***app.use*** function call on route **'/test'** attaches the **testController** router with this route. Now whatever requests our app gets at the '/test', will be handled by our `test-controller.js` router. The '**/anonymous**', '**/user**', '**/admin**', '**/all-user**' route in `test-controller.js` is actually a subroute of '**/test**'.
- Now Check in postman http://localhost:3000/test/anonymous,'http://localhost:3000/test/user',etc...
- As you see all APIs don't require any authentication or authorization. Now let's try to secure these API endpoints.

Preparing Node.js Application to integrate with Keycloak

To configure Node.js application with Keycloak follow the below steps.

1. Install `keycloak-connect`, `express-session` dependencies to your project.

```
npm install keycloak-connect --save
npm install express-session --save
```

2. Create a new folder **`config`** and create a new file **`keycloak-config.js`** in the created folder with the below content. Change the `keycloakConfig` variable content with Keycloak Server URL and `nodejs-microservice` Client Id.

```
var session = require('express-session');
var Keycloak = require('keycloak-connect');

let _keycloak;

var keycloakConfig = {
    clientId: 'nodejs-microservice',
    bearerOnly: true,
    serverUrl: 'http://localhost:8080/auth',
    realm: 'Demo-Realm',
    credentials: {
        secret: '62c99f7c-da55-48fb-ae4e-a27f132546b7'
    }
};
```

```
15
16  function initKeycloak() {
17      if (_keycloak) {
18          console.warn("Trying to init Keycloak again!");
19          return _keycloak;
20      }
21      else {
22          console.log("Initializing Keycloak...");
23          var memoryStore = new session.MemoryStore();
24          _keycloak = new Keycloak({ store: memoryStore },
    keycloakConfig);
25          return _keycloak;
26      }
27  }
28
29  function getKeycloak() {
30      if (!_keycloak){
31          console.error('Keycloak has not been initialized. Please
    called init first.');
32      }
33      return _keycloak;
34  }
35
36  module.exports = {
37      initKeycloak,
38      getKeycloak
39  };
```

Small Note: If the client `Access Type` is `bearer-only` instead of `credentials` you need to provide `realmPublicKey`. Realm Public Key can be copied from Realm Settings > Keys > Public Key.

```
1   var keycloakConfig = {
2       clientId: 'nodejs-microservice',
3       bearerOnly: true,
4       serverUrl: 'http://localhost:8080/auth',
5       realm: 'Demo-Realm',
6       realmPublicKey: 'MIIBIjANBgkqhkiG9w0BAQEFAAO...'
7   };
```

It's the time to initialize Keycloak in **index.js**
Add below lines to index.js after `var app = express();`

```
const keycloak = require('./config/keycloak-config.js').initKeycloak();
app.use(keycloak.middleware());
```

After this change, `index.js` would look like below.

```
var express = require('express');
var app = express();

const keycloak = require('./config/keycloak-config.js').initKeycloak();
app.use(keycloak.middleware());

const testController = require('./controller/test-controller.js');
app.use('/test', testController);

app.get('/', function(req, res){
   res.send("Server is up!");
});

app.listen(4000);
```

Implement Role-based Access for APIs

`keycloak.protect()` Express Request Handler can be used to secure APIs in the routers.
Add the following line to `test-controller.js` to access Keycloak instance initiated in `index.js`

```
const keycloak = require('../config/keycloak-config.js').getKeycloak();
```

/test/anonymous:
This API should be accessible without any Authorization token with no restrictions. It already meets our requirements and needs no additional changes.
/test/user:

This API should be accessible to users with `nodejs-microservice` `user` role. This can be defined by changing the code as below.

```
router.get('/user', keycloak.protect('user'), function(req, res)
{
    res.send("Hello User");
});
```

/test/admin:
This API should be accessible to users with `nodejs-microservice` `admin` role. This can be defined by changing the code as below.

```
router.get('/admin', keycloak.protect('admin'), function(req,
res){
    res.send("Hello Admin");
});
```

/test/all-user:
This API should be accessible to users with `nodejs-microservice` `user` & `admin` roles. This can be defined by changing the code as below.

```
router.get('/all-user', keycloak.protect(['user','admin']),
function(req, res){
    res.send("Hello All User");
});
```

Now Test Controller would look like below.

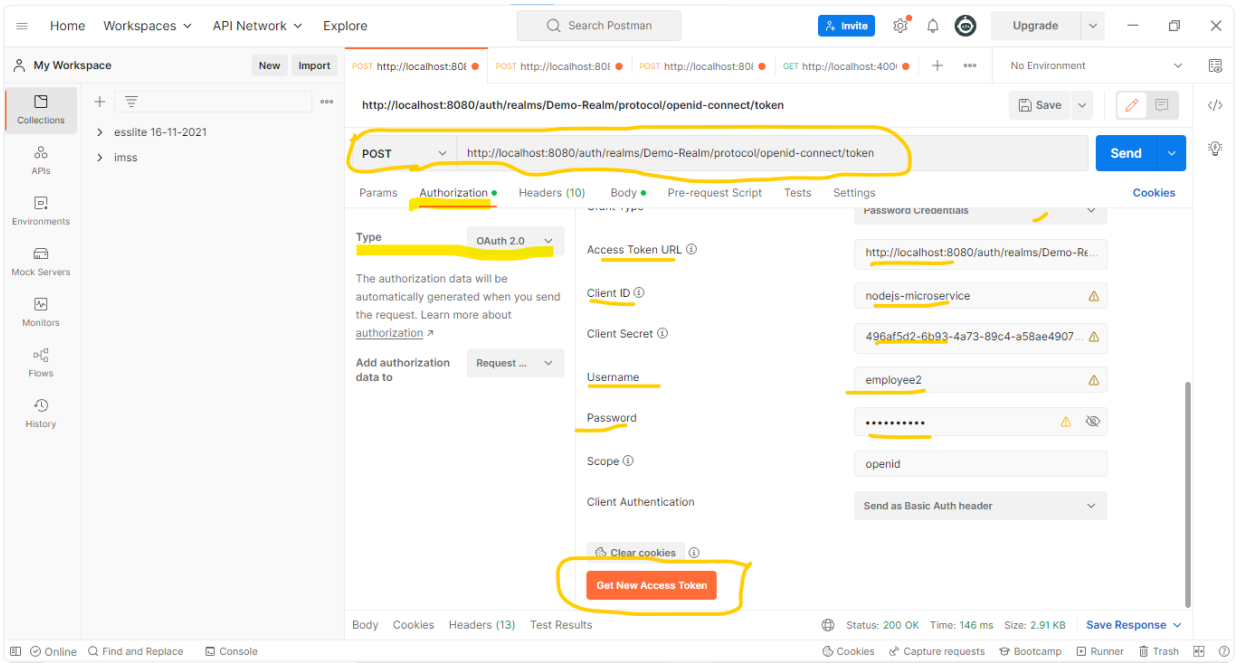
```
var express = require('express');
var router = express.Router();
const keycloak = require('../config/keycloak-
config.js').getKeycloak();

router.get('/anonymous', function(req, res){
    res.send("Hello Anonymous");
});

router.get('/user', keycloak.protect('user'), function(req, res)
{
    res.send("Hello User");
});
```

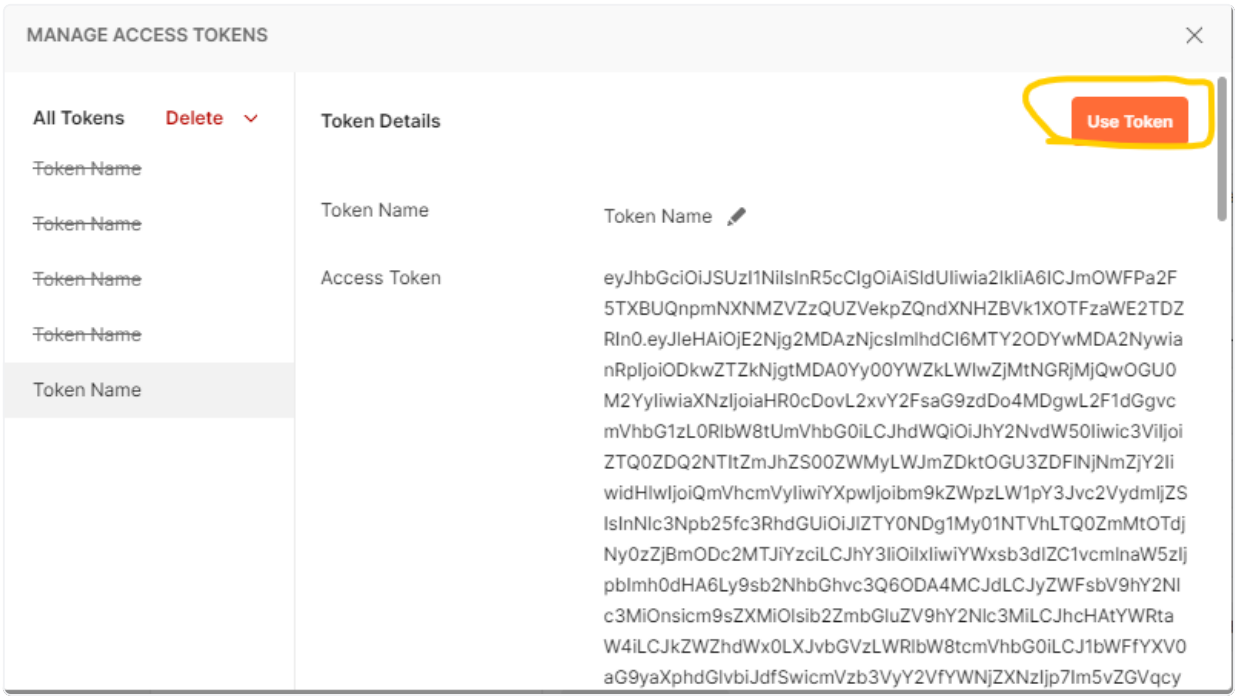
```
12
13  router.get('/admin', keycloak.protect('admin'), function(req,
    res){
14      res.send("Hello Admin");
15  });
16
17  router.get('/all-user', keycloak.protect(['user','admin']),
    function(req, res){
18      res.send("Hello All User");
19  });
20
21  module.exports = router;
```

- Test the above APIs by passing tokens from `employee1`, `employee2`, `employee3` access tokens in the `Authorization` header with the `bearer` prefix (`bearer <ACCESS_TOKEN>`).
  - If the token is expired, you will receive a 401 Unauthorized error.

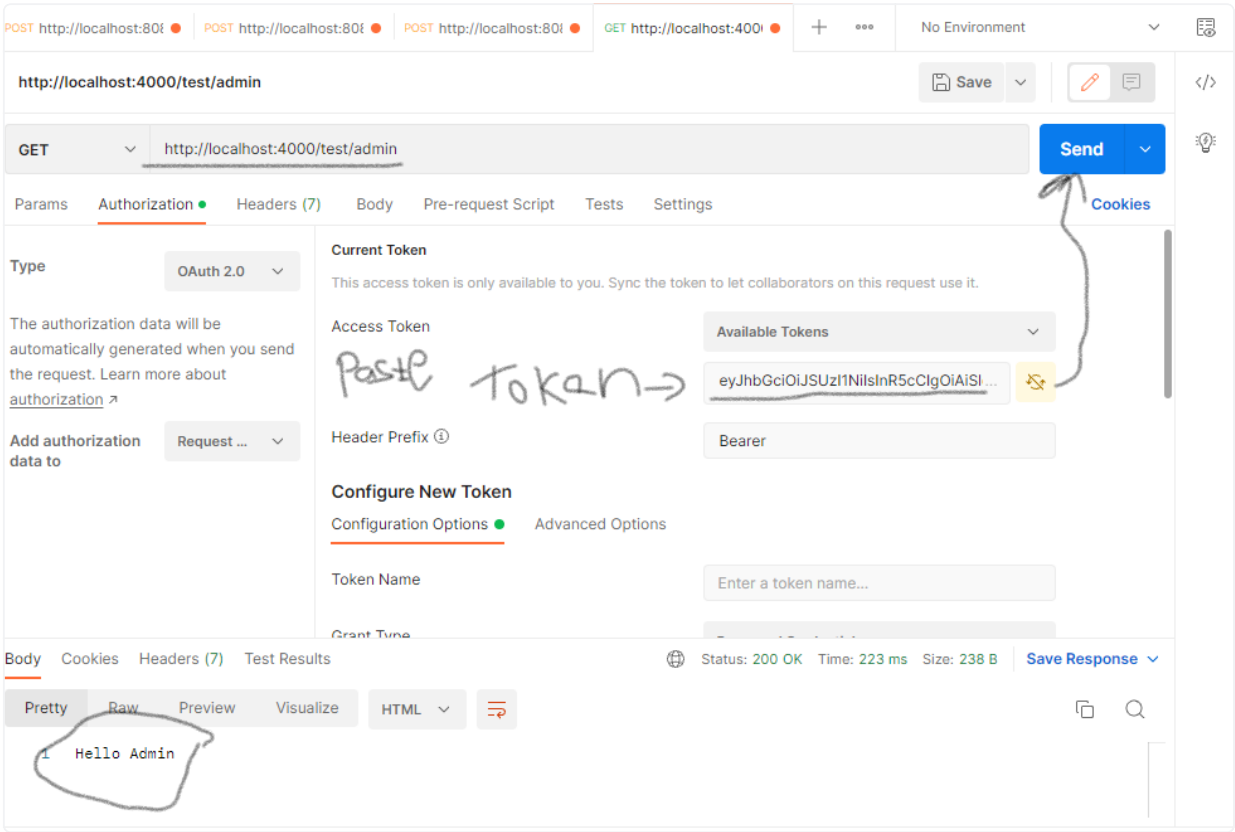
IF the above-generated token not working then follow the below steps to generate a token in another way for better work. Because for me below steps will worked.

Open your Postman And follow the below screens





Click on use Token Then open new get req type your endpoint and paste the Token and click on send you will get the response as below.



Final output

- Thanks....