

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

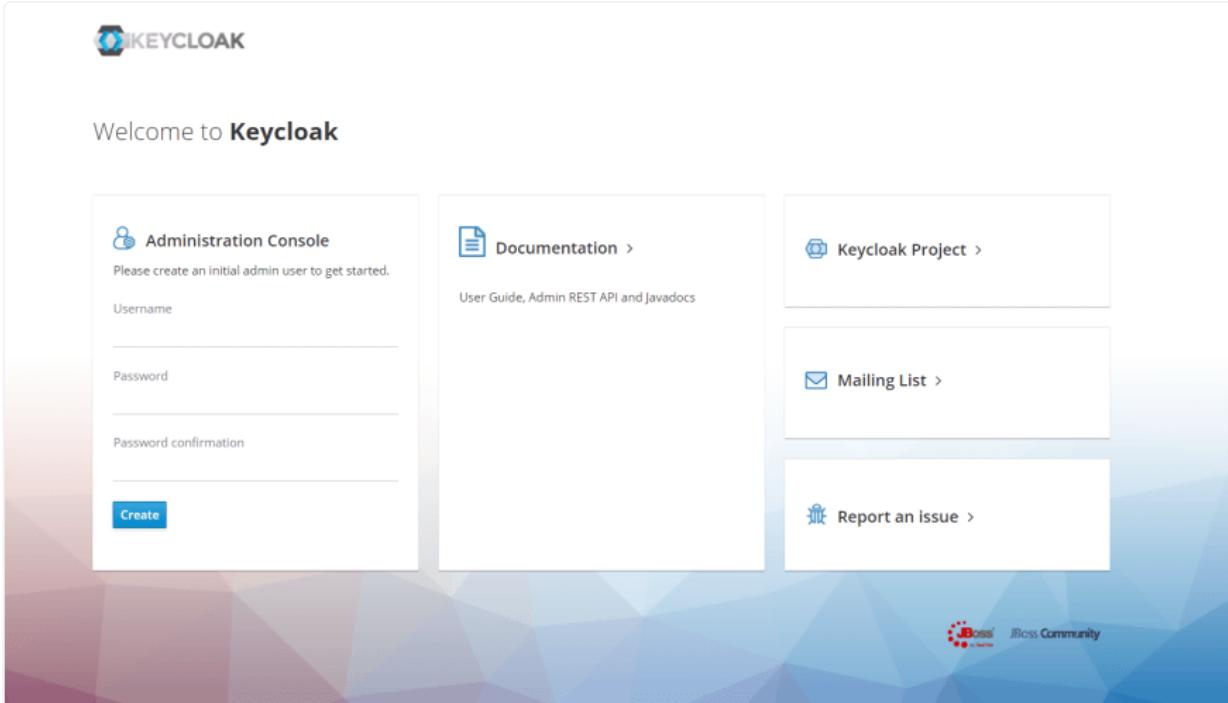
- Download the latest version of keycloak from [here](#).
- Take the zip file
- At the time of writing this, I am using keycloak 15.0.2

If java is not installed,

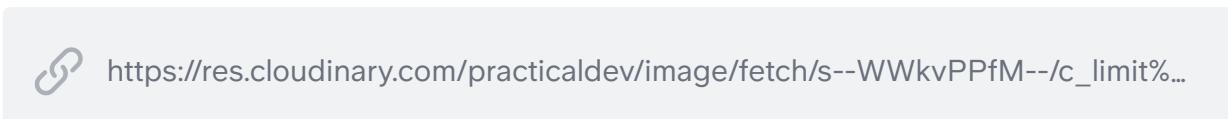
- Install Java SDK
- Add the path to environmental variables
 - After Installing the zip file you should unzip it. If you are using the Linux server you should use this command to unzip `unzip keycloak-15.0.2.zip`.
 - After unzipping go to the bin folder and click on the `standalone.bat` file. you will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

```
C:\Users\fayazbasha\Desktop\keycloak\keycloak-15.0.2\bin>standalone.bat
Calling "C:\Users\fayazbasha\Desktop\keycloak\keycloak-15.0.2\bin\standalone.conf.bat"
Setting JAVA property to "C:\Program Files\Java\jdk-11.0.17\bin\java"
=====
JBoss Bootstrap Environment
JBoss_HOME: "C:\Users\fayazbasha\Desktop\keycloak\keycloak-15.0.2"
JAVA: "C:\Program Files\Java\jdk-11.0.17\bin\java"
JAVA_OPTS: "-Dprogram.name=standalone.bat -Xms64M -Xmx512M -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256M -Djava.net.preferIPv4Stack=true -Djboss.modules.system.pkgs=org.jboss.byteman -Djava.awt.headless=true --add-exports=java.base/sun.nio.ch=ALL-UNNAMED --add-exports=java.base/sun.misc=ALL-UNNAMED --add-exports=java.desktop/sun.reflect=ALL-UNNAMED"
=====
12:34:07,597 INFO [org.jboss.modules] (main) JBoss Modules version 1.11.0.Final
12:34:11,207 INFO [org.jboss.msc] (main) JBoss MSC version 1.4.12.Final
12:34:11,225 INFO [org.jboss.threads] (main) JBoss Threads version 2.4.0.Final
12:34:11,512 INFO [org.jboss.as] (MSC service thread 1-2) WFLYSRV0049: Keycloak 15.0.2 (WildFly Core 15.0.1.Final) starting
12:34:15,604 INFO [org.wildfly.security] (ServerService Thread Pool -- 20) ELY00001: WildFly Elytron version 1.15.3.Final
12:34:18,147 INFO [org.jboss.as.controller.management-deprecated] (ServerService Thread Pool -- 18) WFLYCTL0033: Extension 'security' is deprecated and may not be supported in future versions
12:34:19,570 INFO [org.jboss.as.controller.management-deprecated] (Controller Boot Thread) WFLYCTL0028: Attribute 'security-realm' in the resource at address '/core-service=management/management-interface=http-interface' is deprecated, and may be removed in a future version. See the attribute description in the output of the read-resource-description operation to learn more about the deprecation.
12:34:19,617 INFO [org.jboss.as.controller.management-deprecated] (ServerService Thread Pool -- 11) WFLYCTL0028: Attribute 'security-realm' in the resource at address '/subsystem=undertow/server=default-server/https-listener=https' is deprecated, and may be removed in a future version. See the attribute description in the output of the read-resource-description operation to learn more about the deprecation.
12:34:20,004 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using socket-binding (management-http)
12:34:20,004 INFO [org.jboss.as.server] (Controller Boot Thread) WFLYSRV0039: Creating http management service using socket-binding (management-http)
```

- 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



- 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.
- Type `Demo-Realm` in the **Name** field and click **Create**.

Select realm

Master

Add realm

Import Select file

Name * Demo-Realm

Enabled ON

Create Cancel

- 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.

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

Demo-Realm

Clients

Configure

Realm Settings

Clients

Client Scopes

Roles

Identity Providers

User Federation

Authentication

Client ID	Enabled	Base URL	Actions		
account	True	http://localhost:8080/auth/realm/Demo-Realm/account/	Edit	Export	Delete
account-console	True	http://localhost:8080/auth/realm/Demo-Realm/account/	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
realm-management	True	Not defined	Edit	Export	Delete
security-admin-console	True	http://localhost:8080/auth/admin/Demo-Realm/console/	Edit	Export	Delete

Client Management in Keycloak Admin Console

- 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.

The screenshot shows the Keycloak Admin Console interface. On the left, there is a sidebar with a navigation menu. The 'Clients' option is selected, indicated by a blue highlight. Other options include 'Realm Settings', 'C� Clients', 'Client Scopes', 'Roles', 'Identity Providers', and 'User Federation'. The main content area is titled 'Add Client'. It contains several input fields: 'Import' (with a 'Select file' button), 'Client ID' (set to 'nodejs-microservice'), 'Client Protocol' (set to 'openid-connect'), and 'Root URL' (set to 'http://localhost:8000'). At the bottom right are 'Save' and 'Cancel' buttons. The top navigation bar shows 'Clients > Add Client'.

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**.

The screenshot shows the Keycloak Admin UI for a realm named "Demo-Realm". The left sidebar is titled "Configure" and includes sections for "Realm Settings", "Clients", "Client Scopes", "Roles", "Identity Providers", "User Federation", and "Authentication". Under "Clients", the "Clients" section is selected, showing a list of clients. One client, "nodejs-microservice", is selected and its configuration page is displayed on the right.

The configuration page for "nodejs-microservice" has the following settings:

- Client ID:** nodejs-microservice
- Name:** (empty)
- Description:** (empty)
- Enabled:** ON
- Consent Required:** OFF
- Login Theme:** (empty)
- Client Protocol:** openid-connect
- Access Type:** confidential (selected)
- Standard Flow Enabled:** ON
- Implicit Flow Enabled:** OFF
- Direct Access Grants Enabled:** ON
- Service Accounts Enabled:** ON
- Authorization Enabled:** ON
- Root URL:** http://localhost:8000
- * Valid Redirect URIs:** http://localhost:8000/*

Configure client with Access Type: 'confidential'

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

The screenshot shows the Keycloak interface for managing clients. On the left, a sidebar menu is visible with options like 'Demo-Realm', 'Configure', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. The 'Clients' option is selected. In the main content area, the path 'Clients > nodejs-microservice' is shown. The client name 'Nodejs-microservice' is displayed with a trash can icon. Below the client name, there are tabs: 'Settings' (selected), 'Credentials' (highlighted in blue), 'Roles', 'Client Scopes', 'Mappers', 'Scope', 'Authorization', and 'Revocation'. Under the 'Credentials' tab, there are sections for 'Client Authenticator' (set to 'Client Id and Secret') and 'Secret' (containing the value '62c99f7c-da55-48fb-ae4e-a27f13254'), with a 'Regenerate Secret' button. Another section for 'Registration access token' is also present with a 'Regenerate registration access token' button.

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.

The screenshot shows the 'nodejs-microservice' client roles page. The sidebar menu is identical to the previous screenshot. The main content area shows the client name 'Nodejs-microservice' with a trash can icon. Below the client name, there are tabs: 'Settings', 'Credentials' (selected), 'Roles' (highlighted in blue), 'Revocation', 'Clustering', and 'Installation'. Under the 'Roles' tab, there is a search bar with 'Search...', a magnifying glass icon, and a 'View all roles' button. A message 'No client roles available' is displayed. A 'Add Role' button is located in the bottom right corner of the roles section.

'nodejs-microservice' Client Roles

The screenshot shows the Keycloak interface for adding a new role. The left sidebar is titled 'Demo-Realm' and includes options like 'Configure', 'Realm Settings', 'Clients', 'Client Scopes', 'Roles' (which is selected), 'Identity Providers', and 'User Federation'. The main content area shows the path 'Clients > nodejs-microservice > Roles > Add Role'. The 'Add Role' form has 'Role Name *' set to 'user' and 'Description' set to 'To authorize user permissions'. There are 'Save' and 'Cancel' buttons at the bottom.

Add 'user' role and Save

This screenshot is similar to the previous one, showing the 'Add Role' dialog for the 'admin' role. The 'Role Name *' field contains 'admin' and the 'Description' field contains 'To authorize admin permissions'. The 'Save' and 'Cancel' buttons are visible at the bottom.

Add 'admin' role and Save

The screenshot shows the 'nodejs-microservice' client roles. The left sidebar is identical to the previous screenshots. The main content area shows the 'Roles' tab for the 'nodejs-microservice' client. A table lists two roles: 'admin' and 'user'. The 'admin' role is described as 'To authorize admin permissions' and is marked as 'Composite' (False). The 'user' role is described as 'To authorize user permissions' and is also marked as 'Composite' (False). There are 'Edit' and 'Delete' actions for each role.

Role Name	Composite	Description	Actions
admin	False	To authorize admin permissions	Edit Delete
user	False	To authorize user permissions	Edit Delete

'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.

Role Name	Composite	Description	Actions
offline_access	False	#{role_offline-access}	Edit Delete
uma_authorization	False	#{role_uma_authorization}	Edit Delete

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**.

* Role Name: app-user
Description: Application user permissions

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.

The screenshot shows the Keycloak admin interface for the 'Demo-Realm'. The left sidebar is collapsed, and the main area shows the 'app-admin' role configuration. The 'Details' tab is active, displaying the role's name ('app-admin') and description ('Application admin permissions'). Below this, the 'Composite Roles' section is expanded, showing two tabs: 'Realm Roles' and 'Client Roles'. Under 'Realm Roles', there is a list of available roles: 'app-user', 'offline_access', and 'uma_authorization'. Under 'Client Roles', there is a list of available roles: 'user'. In the 'Associated Roles' section, the role 'admin' is listed and highlighted with a grey background.

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.

The screenshot shows the Keycloak administration interface for a 'Demo-Realm'. The left sidebar has a dark theme with white text. Under 'Configure', 'Realm Settings' is listed. Under 'Manage', 'Groups', 'Users' (which is selected and highlighted in blue), 'Sessions', 'Events', and 'Import' are listed. The main content area is titled 'Add user'. It contains fields for 'ID' (empty), 'Created At' (empty), 'Username *' (containing 'employee1'), 'Email' (empty), 'First Name' (empty), 'Last Name' (empty), 'User Enabled' (set to 'ON'), 'Email Verified' (set to 'ON'), and a dropdown for 'Required User Actions' with the placeholder 'Select an action...'. At the bottom are 'Save' and 'Cancel' buttons. The overall title of the window is 'Users > Add user'.

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.

The screenshot shows the Keycloak 'Users' page for the 'Demo-Realm'. The left sidebar is titled 'Demo-Realm' and contains sections for 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users, Sessions). The 'Users' section is currently selected. The main content area is titled 'Employee1' and shows the 'Credentials' tab. Below it is a table titled 'Manage Credentials' with columns for Position, Type, User Label, Data, and Actions. A form for setting a password is present, with fields for 'Password' and 'Password Confirmation' both containing '*****', a 'Temporary' switch set to 'OFF', and a 'Set Password' button.

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.

The screenshot shows the 'Role Mappings' tab for 'Employee1'. The left sidebar is identical to the previous screenshot. The main content area shows the 'Role Mappings' tab selected. It features four panels: 'Realm Roles' (containing 'app-admin'), 'Available Roles' (containing 'app-admin'), 'Assigned Roles' (containing 'app-user', 'offline_access', and 'uma_authorization'), and 'Effective Roles' (containing 'app-user', 'offline_access', and 'uma_authorization'). Buttons for 'Add selected >' and 'Remove selected <' are visible between the 'Available Roles' and 'Assigned Roles' panels.

`employee1` Role Assignment

The screenshot shows the Keycloak interface for the 'Demo-Realm'. On the left, a sidebar lists 'Configure' options: Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, and Authentication. The main area shows a user named 'employee2'. The 'Role Mappings' tab is selected. Under 'Realm Roles', there is one entry: 'Available Roles' with 'app-user'. Under 'Assigned Roles', there are three entries: 'app-admin', 'offline_access', and 'uma_authorization'. Under 'Effective Roles', there are three entries: 'app-admin', 'offline_access', and 'uma_authorization'. Buttons for 'Add selected' and 'Remove selected' are visible between the 'Available Roles' and 'Assigned Roles' sections.

`employee2` Role Assignment

This screenshot is similar to the previous one, showing the Keycloak interface for 'employee3'. The 'Role Mappings' tab is selected. Under 'Available Roles', there is one entry: 'app-user'. Under 'Assigned Roles', there are four entries: 'app-admin', 'app-user', 'offline_access', and 'uma_authorization'. Under 'Effective Roles', there are four entries: 'app-admin', 'app-user', 'offline_access', and 'uma_authorization'. The layout is identical to the 'employee2' screenshot, with 'Add selected' and 'Remove selected' buttons.

`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.

Demo-Realm

Configure

Realm Settings

Clients

Client Scopes

Roles

Identity Providers

User Federation

Authentication

Manage

Groups

Users

Sessions

General Login Keys Email Themes Cache Tokens Client Registration Security Defenses

* Name: Demo-Realm

Display name:

HTML Display name:

Frontend URL:

Enabled: ON

User-Managed Access: OFF

Endpoints:

- OpenID Endpoint Configuration
- SAML 2.0 Identity Provider Metadata

Save Cancel

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:

```
1 <KEYCLOAK_SERVER_URL>/auth/realms/<REALM_NAME>/protocol/openid-
connect/tokenEx:
2   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.

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

Example:

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

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

POST http://localhost:8080/auth/realms/Demo-Realm/protocol/openid-connect/token Send Save

Params Authorization Headers (9) Body **Pre-request Script Tests Settings Cookies Code**

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> grant_type	password			
<input checked="" type="checkbox"/> client_id	nodejs-microservice			
<input checked="" type="checkbox"/> client_secret	62c99f7c-da55-48fb-ae4e-a27f132546b7			
<input checked="" type="checkbox"/> username	employee1			
<input checked="" type="checkbox"/> password	mypassword			
Key	Value	Description		

Body Cookies (3) Headers (8) Test Results Status: 200 OK Time: 94 ms Size: 2.62 KB Save Response

Pretty Raw Preview Visualize JSON  

```
1 "access_token":  
2 "eyJhbGciOiJSUzIiNiIsInR5cC IgOiAiSldUiwi a2lkiA6ICJNRm5jZHZheVhsVXZlM2dBMVoyNk80bFVKR3Rr0XFxn2cwc0VLUF90R1hnIn0.  
eyJleHAiOjE10Dc3NzALMTcsImIhdCI6MTU4Nzc3MDExNyvianRpIjo1MGU2MjBjYmIt0gJLNC0Zjg1LTlMDUtDFhZdkwNTIzzJzUiWnXzIjoi  
HR0cDovL2xvY2FsaG9zD04MDgwL2F1dGgvcmVhbG1z0Rlbw8tUmVhbG0iLCJhdwQio1sC3byaw5nY9vdC1taWnby3nLcnZpY2uiLCJhY2NvdW50Il  
0sInN1YiI6jEyTc4YTQ2LTlhMjgtNDQ0MS040DQ1LWFkMDg4YmU2ZGU1MCIsInR5cCI61k1lyXJlcisImF6cCI6Im5vZGvqcy1taWnby3nLcnZpY2U  
iLCJzZXNzaWu9uX3N0YXRlIjoiNGQzMa50WQtNmIz0C0080mEwLWl3YwQ0Yj5NmYhMgQyND13IiwiYm0iMSisImFsbG93ZWtb3jZ2Lucy16Wyo  
dHRw0i8vbG9jYWxbob3N0jgwmDAiXSwicmVhbG1fYWnjZXNzIjp7InJvbGVzIjpbiM9mZmxbmVfyWnjZXNzIiwidWlhX2F1dGhvcmI6XRpb24iLCJhc  
HAtdxN1ciJdfSwicmVzb3Vyy2VFYmnjZXNzIjp7InNwcmLuZ2Jzb30tbwljcm9zZXj2aWnLIjp7InJvbGVzIjpbiM9mZmxbmVfyWnjZXNzIiwidWlhX2F1dGhvcmI6XRpb24iLCJhc  
NlcnPzY2Ui0nsicmsZXM10lsidXNlcjDfSwiYWnjb3VudCI6jeYjb2xlcI6WYjtYWh5z2UtyWnjb3VudCisIm1hbmfnsZs1hY2Nvdw50LwpxpmtzIi  
idmlldy1wcm9maWxL1I19fSwic2NvcGUi0jwcm9maWxL1GVTyWl1siwiwZ1haWxfdmVyaZpZWQioNrydWusInByZWlcnJlZf91c2VybmtZs16Imvt  
cGxveWlMSj9.  
A-wB_Eab5h3RqWdBuSzQHrru0AvlmQo2wCB9X9V9AuTJRJ5omTGHj0IFGj3zLlpKEzygCtnk8xulc0kj5fn3E_QC62IP-bcDbwk7_uZ2tdv7hZ2qqLZE  
Nu2yWDvTiZIFZFaHwmx9MeUxdU76kwF3vbP0LtnxqCmFT0qaGqWb6xkEHcOh0Mihx6VJRFWGNppiq-6v5uCqtcZEzy57HJ0QWxcxqWBMA5q5L2mEejml  
L-Et_aBuTPt2wtf74T8_kuB44qejMBVqdWakEI9Bh32V_iTMDLeq-YjIazvNINVzDmVFsma5x0kL1RzXj3xhYQVbE6ggxqmYNNtDy3nnmQQ",  
3 "expires_in": 300,  
4 "refresh_expires_in": 1800,  
5 "refresh_token":  
"eyJhbGciOiJIUzIiNiIsInR5cC IgOiAiSldUiwi a2lkiA6ICJiMjA0MTRhZi1kMmE2LTQzXTiTjYfhZc0xY2Uy0GjY2F10DYifQ.eyJleHAiOjE10  
Dc3NzEMTcsImIhdCI6MTU4Nzc3MDExNyvianRpIjo1YTk2GNLZG0t0DEwMj00M20xLTg4ZTQtMmY2ZTNjN2Vj0DQ1IiwiXzIjoiAhR0cDovL2xvY2  
FsaG9zD04MDgwL2F1dGgvcmVhbG1z0Rlbw8tUmVhbG0iLCJhdwQio1jodHRw0i8vbG9jYwxbob3N0jgwoDAvYXV0aC9yZWFsbXVmRGVtby1sZWFsbSI  
sInN1YiI6jEyTc4YTQ2LTlhMjgtNDQ0MS040DQ1LWFkMDg4YmU2ZGU1MCIsInR5cCI61ljlZnJlc2giLCJhenAi0iJub2RlanMtbWljcm9zZXj2aWnL  
IiwicVz2lvbl9zDf0ZSi6jIjRkmzMoWt1kLTzIzmgtnghmc1iN2fkLWl50TzmjBkMjQyNyIsInNjbj3BLijoiChjVzmlsZsLBwFpbCj9.  
I23V-yBcRikM9Vz8Pvdl9lkLku5GJKTHpkvF4qe3Dk",  
6 "token_type": "bearer",  
7 "not-before-policy": 0,  
8 "session_state": "4d33099d-6b38-4ba0-b7ad-b996f20d2427",  
9 "scope": "profile email"  
10 }
```

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`)

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldeUiwi  
a2lkIiA6ICJNRm5JZHZeVhsVXZlM2dBMVoyNk80  
bFVKR3Rr0XFxn2cwc0VLUF90R1hnIn0.eyJleHAI  
0jE10Dc3NzA0MTcsImlhcdI6MTU4Nzc3MDExNywi  
anRpIjoiMGU2MjBjYmItOGJ1NC00Zjg1LTlkMDUt  
MDFhZDkwNTIzZjUwIiwiXNzIjoiaHR0cDovL2xv  
Y2FsaG9zdDo4MDgwL2F1dGvcmVhbG1zL0R1bW8t  
UmVhbG0iLCJhdWQiolsic3ByaW5nYm9vdC1taWNy  
b3NlcnPzY2UiLCJhY2NvdW50Il0sInN1YiI6IjEy  
YTc4YTQ2LTlhMjgtNDQ0MS040DQ1LWFkMDg4YmU2  
ZGU1MCIsInR5cCI6IkJlYXJlcIisImF6cCI6Im5v  
ZGVqcy1taWNyb3NlcnPzY2UiLCJzZXNzaW9uX3N0  
YXR1IjoiNGQzMzA50WQtNmIz0C00YmEwLWI3YWQt  
Yjk5NmYyMGQyNDI3IiwiYWNNyIjoiMSIsImFsbG93  
ZWQtB3JpZ2lucyI6WyJodHRwOi8vbG9jYWxob3N0  
OjgwMDAiXSwicmVhbG1fYWNjZXNzIjp7InJvbGVz  
Ijpblm9mZmxpbmVfYWNjZXNzIiwidW1hX2F1dGhv  
cm16YXRpb24iLCJhcAtxdXNlciJdfSwicmVzb3Vy  
Y2VfYWNjZXNzIjp7InNwcmLuZ2Jvb3QtbWljcm9z  
ZXJ2aWNlIjp7InJvbGVzIjpblm9mZmxpbmVfYWNjZXNzIiwidW1hX2F1dGhv  
ZGVqcy1taWNyb3NlcnPzY2UiOnsicm9sZXMiolsi  
dXNlciJdfSwiYWNjb3VudCI6eyJyb2xlcyI6WyJt  
YW5hZ2UtYWNjb3VudCIisIm1hbmfNzS1hY2NvdW50  
LWxpbtzIiwiidmlldy1wcm9maWx1Il19fSwic2Nv  
cGUiOjJwcm9maWx1IGVtYWlsIiwiZW1haWxfdmVy  
aWZpZWQiOnRydWUsInByZWZlcnJlZF91c2VybmfT  
ZSI6ImVtcGxveWV1MSJ9.A-  
wB_Eab5h3RqWdBuSZQHrru0AvlmQzAwCB9X9VAu  
TJRJ5omTghJ0IFGi3zL1pKEzygCtn8xulcQkj5  
fn3E_QC62IP-  
bcDwbk7_uZ2tdv7hZ2qq1ZENu2yWDvTiFZIFZfaH  
Wmx9MEuXdu76kwF3vbPOLtNXqCmFT0qaGqWb6xkE  
Hc0h0Mihx6VJRFGNppiq-  
6v5uCqtcZEzy57HJ0QWxcxqWBMA5q5L2mEejmLL-  
Et_aBUTPt2wtf74T8_kuB44qejMBVQdWakEI9Bh3  
2V_iTMDeq-  
Yj1AZvVNINVzDmVFSma5x0kL1RzXJ3xhYQVbEGg  
xqmYNNtDy3nnmQQ
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  
  "alg": "RS256",  
  "typ": "JWT",  
  "kid": "MFNldvayXlUve3gA1Z26041UJGt9qW7g0sEKP_tGXg"  
}
```

PAYLOAD: DATA

```
{  
  "exp": 1587770417,  
  "iat": 1587770117,  
  "jti": "0e620ccb-8be4-4f85-9d05-01ad98523f50",  
  "iss": "http://localhost:8080/auth/realm/Demo-Realm",  
  "aud": [  
    "springboot-microservice",  
    "account"  
  ],  
  "sub": "12a78a46-9a28-4441-8845-ad088be6de50",  
  "typ": "Bearer",  
  "azp": "nodejs-microservice",  
  "session_state": "4d33099d-6b38-4ba0-b7ad-b996f20d2427",  
  "acr": "1",  
  "allowed_origins": [  
    "http://localhost:8080"  
  ],  
  "realm_access": {  
    "roles": [  
      "offline_access",  
      "uma_authorization",  
      "app-user" ←  
    ]  
  },  
  "resource_access": {  
    "springboot-microservice": {  
      "roles": [  
        "user"  
      ]  
    },  
    "nodejs-microservice": {  
      "roles": [  
        "user" ←  
      ]  
    },  
    "account": {  
      "roles": [  
        "manage-account",  
        "manage-account-links",  
        "view-profile"  
      ]  
    },  
    "scope": "profile email",  
    "email_verified": true,  
    "preferred_username": "employee1"  
  }  
}
```

VERIFY SIGNATURE

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.

{

```
"exp": 1587770417, ← Sat Apr 25 2020 00:20:17 GMT+0100 (British Summer Time)  
"iat": 1587770117,
```

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.

```

1 var express = require('express');
2 var app = express();
3
4 app.get('/', function(req, res){
5     res.send("Server is up!");
6 });
7
8 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.

```

1 var express = require('express');
2 var router = express.Router();
3
4 router.get('/anonymous', function(req, res){
5     res.send("Hello Anonymous");
6 });
7 router.get('/user', function(req, res){
8     res.send("Hello User");
9 });
10
11 router.get('/admin', function(req, res){
12     res.send("Hello Admin");
13 });
14
15 router.get('/all-user', function(req, res){
16     res.send("Hello All User");
17 });
18
19 module.exports = router;

```

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

```

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

```
2 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 subroutine 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.

```
1 npm install keycloak-connect --save
2 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.

```
1 var session = require('express-session');
2 var Keycloak = require('keycloak-connect');
3
4 let _keycloak;
5
6 var keycloakConfig = {
7   clientId: 'nodejs-microservice',
8   bearerOnly: true,
9   serverUrl: 'http://localhost:8080/auth',
10  realm: 'Demo-Realm',
11  credentials: {
12    secret: '62c99f7c-da55-48fb-ae4e-a27f132546b7'
13  }
14};
```

```

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 },
25          keycloakConfig);
26          return _keycloak;
27      }
28
29  function getKeycloak() {
30      if (!_keycloak){
31          console.error('Keycloak has not been initialized. Please
32          called init first.');
33      }
34      return _keycloak;
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();`

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

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

```
1 var express = require('express');
2 var app = express();
3
4 const keycloak = require('./config/keycloak-
  config.js').initKeycloak();
5 app.use(keycloak.middleware());
6
7 const testController = require('./controller/test-
  controller.js');
8 app.use('/test', testController);
9
10 app.get('/', function(req, res){
11   res.send("Server is up!");
12 });
13
14 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`

```
1 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.

```
1  router.get('/user', keycloak.protect('user'), function(req, res){  
2      res.send("Hello User");  
3  });
```

/test/admin:

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

```
1  router.get('/admin', keycloak.protect('admin'), function(req, res){  
2      res.send("Hello Admin");  
3  });
```

/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.

```
1  router.get('/all-user', keycloak.protect(['user', 'admin']),  
2  function(req, res){  
3      res.send("Hello All User");  
4  });
```

Now Test Controller would look like below.

```
1  var express = require('express');  
2  var router = express.Router();  
3  const keycloak = require('../config/keycloak-  
config.js').getKeycloak();  
4  
5  router.get('/anonymous', function(req, res){  
6      res.send("Hello Anonymous");  
7  });  
8  
9  router.get('/user', keycloak.protect('user'), function(req, res){  
10     res.send("Hello User");  
11  });
```

```

12
13   router.get('/admin', keycloak.protect('admin'), function(req,
14     res){
15       res.send("Hello Admin");
16   });
17
18   router.get('/all-user', keycloak.protect(['user','admin']),
19   function(req, res){
20     res.send("Hello All User");
21   });
22
23   module.exports = router;

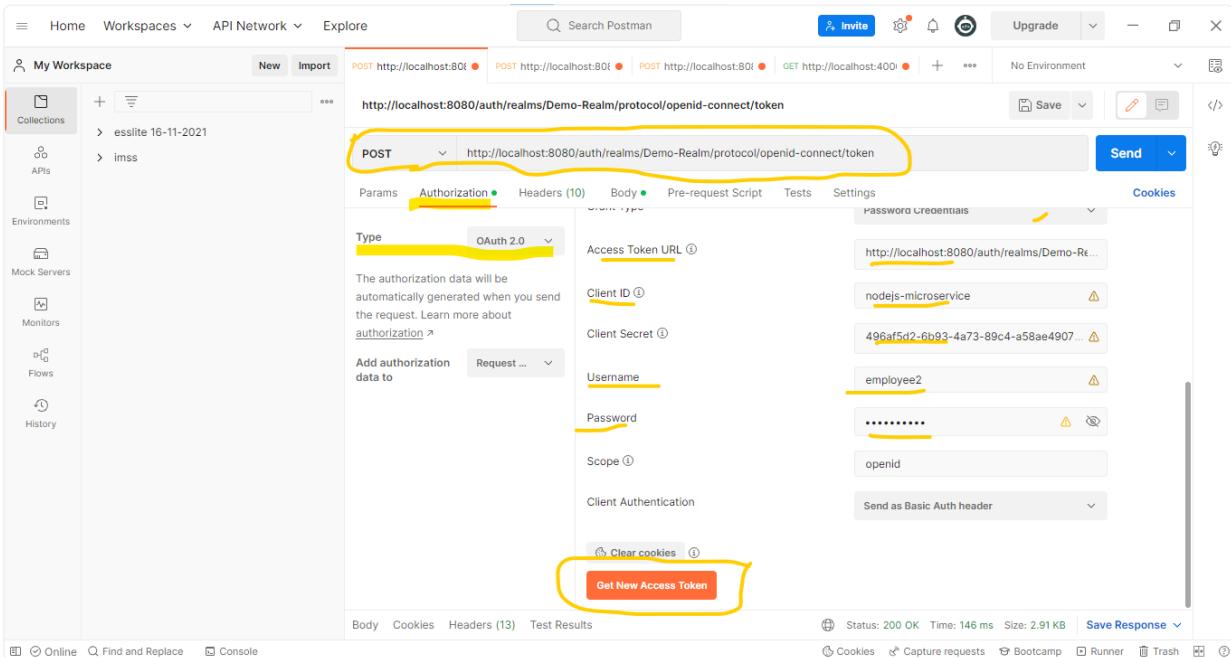
```

- 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



The screenshot shows the Postman interface with the following details:

- Request Method:** POST
- URL:** `http://localhost:8080/auth/realm/Demo-Realm/protocol/openid-connect/token`
- Type:** OAuth 2.0
- Authorization:** (highlighted with yellow)
- Params:** (highlighted with yellow)
- Headers:** (highlighted with yellow)
- Body:** (highlighted with yellow)
- Tests:** (highlighted with yellow)
- Settings:** (highlighted with yellow)
- Cookies:** (highlighted with yellow)
- Authorization Fields:**
 - Access Token URL: `http://localhost:8080/auth/realm/Demo-Realm/protocol/openid-connect/token`
 - Client ID: `nodejs-microservice`
 - Client Secret: `498af5d2-6b93-4a73-89c4-a58ae4907...`
 - Username: `employee2`
 - Password: (redacted)
 - Scope: `openid`
 - Client Authentication: `Send as Basic Auth header`
- Buttons:**
 - Get New Access Token** (highlighted with red)
 - Clear cookies**

MANAGE ACCESS TOKENS

All Tokens	Delete	Token Details
Token Name		Token Name <input type="text"/>
Token Name		Access Token <input type="text"/>
Token Name		eyJhbGciOiJSUzI1NilsInR5cClgOiAiSlUliwia2IkliA6ICJmOWFPa2F5TXBUQnprmNXNMZVZzQUZVekpZQndXNHZBVk1XOTFzaWE2TDZRln0eyJleHAIoJE2Njg2MDAzNjcsImhdCI6MTY2ODYwMDA2NywanRpIjoiODkwTZkNjgtMDAOYy00YWZkLWlwZjMtNGRjMjQwOGU0M2YylwiiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo4MDgwL2F1dGgvcmVhbG1zL0RlbW8tUmVhbG0iLCJhdWQiOjhY2NvdW50liwic3ViljoiZTQ0ZDQ2NTItZmJhZS00ZWMyLWJmZDktOGU3ZDFINjNmZjY2liwidHlwIjoiQmVhcmVylwiYXpwljoibm9kZWpzLW1pY3Jvc2VydmijZSlsInIc3Npb25fc3RhdGUoIjIZTY0NDg1My01NTVhLTQ0ZmMtOTdjNy0zJjBmODc2MTJiYzciLCJhY3liOixliwiYWxsb3dlZC1vcmlnaW5ljpblmh0dHA6Ly9sb2NhbGhv3Q6ODA4MCJdLCJyZWFsbV9hY2NIc3MiOnsicm9sZXMiOlsib2ZmbGluZV9hY2Nlc3MiLCJhcAtYWRtaW4iLCJkZWZhdWx0LXJvbGVzLWRlbW8tcmVhbG0iLCJ1bWFfYXVOaG9yaXphdGlvbiJdfSwicmVzb3VyY2VfYWNjZXNzIjp7Im5vZGVqcy
Token Name		

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.

The screenshot shows the Postman interface with the following details:

- Authorization Tab:** OAuth 2.0 selected. A note says: "The authorization data will be automatically generated when you send the request. Learn more about [authorization](#)".
- Current Token:** A note says: "This access token is only available to you. Sync the token to let collaborators on this request use it."
- Access Token:** A dropdown menu shows "Available Tokens" with the value "eyJhbGciOiJSUzI1NilsInR5cClgOiAiSl...". A handwritten note "Paste Token" points to this field.
- Header Prefix:** Bearer
- Configure New Token:** Configuration Options selected. A note says: "Enter a token name..."
- Test Results:** Status: 200 OK, Time: 223 ms, Size: 238 B. The response body shows "Hello Admin".

Final output

- Thanks....