

# Keycloak: Autorización Centralizada

En esta página vamos a explicar como podemos implementar la estrategia de autorización centralizada para todos nuestro endpoints utilizando la herramienta de Keycloak.

Toda la documentación y gráficos han sido sacados de la documentación oficial de Keycloak, que puede ser accedida desde este [link](#).

## Conceptos básicos

Antes de ver como configurar los clientes de keycloak para poder implementar esta estrategia de autorización, vamos a nombrar explicar primero algunos conceptos básicos que tienen que ver con esta estrategia y que están explicados en la especificación [standard del protocolo UMA 2.0 \(User-Managed Access\)](#). O tambien en la página oficial de Keycloak sobre el servicio que implementa este protocolo llamado [Authorization Service](#).

Voy a centrarme exclusivamente en las entidades que deberemos de configurar en Keycloak para poder controlar el acceso a nuestros endpoints (recursos) desde la plataforma de administrador de Keycloak

- **Resource:** Un resource es parte de los activos de una aplicación y de la organización. Puede ser un conjunto de uno o más endpoints, un recurso web clásico como una página HTML, etc. En la terminología de la política de autorización, un recurso es el objeto que se protege.
- **Scope:** El scope de un resource es una extensión limitada de acceso que es posible realizar en un resource. En la terminología de la política de autorización, un scope es uno de los muchos verbos que pueden aplicarse lógicamente a un resource.
- **Policy:** Una policy define las condiciones que deben cumplirse para otorgar acceso a un objeto. A diferencia de los permisos, no especifica el objeto que se protege, sino las condiciones que deben cumplirse para acceder a un objeto determinado (por ejemplo, resource, scope o ambos).
- **Permission:** Un permission asocia el objeto que se protege con las policies que se deben evaluar para determinar si se otorga el acceso.

Para mayor detalle de estos conceptos consultar la [página oficial de Keycloak](#)

## Ventajas de esta estrategia

- **Ahorro en el desarrollo:** la casuística entorno a la autorización con permisos es muy amplia y compleja de implementar. Keycloak no solo la implementa cumpliendo toda la casuística posible sino que cumple un standard llamado UMA 2.0 al igual que ocurre con la autenticación/autorización OpenId/Outh 2.1. Igualmente todos los servicios implementados en el lenguaje que sea, se aprovecharan de esta implementación manejada y desacoplada de cualquier servicio que la requiera.
- **Dinamismo:** la capacidad de delegar la estrategia de autorización a Keycloak, permite que el control de acceso a nuestros endpoint sea transparente y dinámica, no teniendo que reprogramar y desplegar nuestros servicios de backend.
- **Integración con sistemas externos:** igualmente el standard UMA 2.0 ofrece endpoints que pueden ser consultados por cualquier servicio para recoger todos los roles y permisos asociados a un usuario y validar que este puede acceder a todos los recursos que se le han sido asignados.

## Flujo de trabajo para un caso de uso de ejemplo

En primer lugar vamos a definir un caso de uso o ejemplo através de la siguiente **tabla de permisos** que queremos implementar utilizando la estrategia centralizada de Keycloak. En ella se puede ver

- **Role:** representa el **role** asociado al usuario que quiere acceder a un endpoint (**resource**) concreto del sistema
- **Resource:** representa el endpoint sobre el que queremos aplicar una política de acceso (**Policy**)
- **Scope:** representa bajo que contexto el usuario va a acceder a este recurso
- **Permission:** representa si este usuario con un determinado **role** puede acceder al recurso (**resource**) bajo un ámbito (**scope**)

Role	Resource	Scope	Permission
Admin	Client	GET	SI
Admin	Client	POST	SI
Admin	Client	PUT	SI
Admin	Client	DELETE	SI
Operador	Client	GET	SI
Operador	Client	POST	NO
Operador	Client	PUT	NO
Operador	Client	DELETE	NO
User	Client	GET	SI

User	Client	POST	NO
User	Client	PUT	NO
User	Client	DELETE	NO
Admin	Product	GET	SI
Admin	Product	POST	SI
Admin	Product	PUT	SI
Admin	Product	DELETE	SI
Operador	Product	GET	SI
Operator	Product	POST	SI
Operator	Product	PUT	NO
Operator	Product	DELETE	NO
User	Product	GET	SI
User	Product	POST	NO
User	Product	PUT	NO
User	Product	DELETE	NO

Ahora que hemos definido que **tabla de permisos** queremos implementar, vamos a explicar los pasos a seguir a la hora de implementar esta estrategia de autorización-

### STEP 01: Creación y configuración del cliente

En primer lugar deberemos de crear el **cliente** que implemente esta estrategia de autorización bajo el realm que representa a nuestra Organización. Este cliente ha de cumplir una condición y es que debe ser del tipo **confidential**, pues solo este tipo de cliente permite configurar este tipo de estrategia de autorización.

Una vez definida este tipo de cliente debemos de activar la opción llamada **Authorization Enabled**. Este flag activo hará aparecer una nueva pestaña bajo nuestro cliente llamada **Authorization**, desde donde podremos configurar esta estrategia de autorización para este cliente.

The screenshot shows the Keycloak configuration interface for a client named 'poc-backend'. The 'Authorization' tab is selected and highlighted with a red box. The 'Access Type' dropdown is set to 'confidential' and is also highlighted with a red box. The 'Authorization Enabled' toggle switch is turned on and is highlighted with a red box. Other settings like 'Client ID', 'Name', 'Description', 'Enabled', 'Always Display in Console', 'Consent Required', 'Login Theme', 'Client Protocol', 'Standard Flow Enabled', 'Implicit Flow Enabled', 'Direct Access Grants Enabled', 'Service Accounts Enabled', 'OAuth 2.0 Device Authorization Grant Enabled', 'OIDC CIBA Grant Enabled', 'Front Channel Logout', and 'Root URL' are visible.

## STEP 02: Identificar los recursos a proteger y los scopes para cada uno de ellos

Una vez tenemos creado nuestro cliente, con las configuraciones antes descritas, ya podemos pasar configurar esta estrategia desde esta nueva pestaña habilitada a al efecto.

En primer lugar debemos de definir que recursos de nuestra aplicación y bajo que scopes van a ser utilizados por nuestras indentidades (usuarios, servicios). En este diagrama se ven los pasos a seguir.



En el ejemplo que vamos a mostrar, vamos a configurar esta estrategia de acceso para dos endpoints, que el standard los llama recursos. Un enunciado podría ser este:

“Vamos controlar el acceso a dos endpoints que representan cada uno de ellos a las entidades: Producto y Cliente y sobre los cuales hemos implementado un CRUD en nuestro backend, para poder: visualizar, crear, editar y borrar cada uno de ellos”.

Con este enunciado vamos a configurar esta estrategia de autorización.

En primer lugar nos dice que estos endpoints implementan un CRUD, por lo tanto podemos acceder a los mismos de cuatro formas diferentes, que representan los cuatro verbos de todos servicio RESTFull: GET, PUT, POST y DELETE. Por lo tanto vamos a crear los **scopes**, que como ya hemos explicado en el apartado anterior representan la forma en que queremos acceder a nuestro **resources** (endpoints). Para ello accedemos a la **pestaña Authorization scopes** situada dentro de la **pestaña Authorization** activada. La configuración que queremos implementar puede verse en la siguiente tabla:

scope	verb
view	GET
create	POST
edit	PUT
delete	DELETE

A la hora de crear un scope simplemente le dasmos el nombre antes descrito en la tabla, sabiendo el contexto que tiene cada uno de ellos. Este proceso lo repetimos cuatro veces uno por cada uno de los verbos que nuestros endpoints implementan.

KEYCLOAK

Poc

Configure

- Realm Settings
- Clients
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions
- Events

Clients > poc-backend > Authorization > Scopes > create

### Create

Name: create

Display name: Create

Icon URI:

Save Cancel

Se puede ver en la imagen inferior como hemos creado los cuatro scopes antes citados. Como podemos ver los campos que rellenamos a la hora de crear un role son:

- **Name:** nombre único asociado a nuestro scope. Este **identificador lo utilizaremos** después cuando queramos chequear si un usuario con un rol concreto puede o no acceder a un recurso bajo un scope seleccionado

- **Display name:** breve descripción de nuestro scope, para poder seleccionarlo o listarlo fácilmente.

The screenshot shows the Keycloak Admin Console interface. On the left is a dark sidebar with navigation options like 'Configure', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', 'Authentication', 'Manage', 'Groups', 'Users', 'Sessions', 'Events', 'Import', and 'Export'. The main content area shows the configuration for a client named 'Poc-backend'. The 'Authorization' tab is selected, and within it, the 'Scopes' sub-tab is active. A table lists the following scopes:

Name
> create
> delete
> edit
> view

Ahora que ya tenemos los scopes de nuestros todos los scopes implementados por nuestros dos recursos, vamos a crear estos recursos y asociarles los scopes implementados por cada uno de ellos. Podemos ver en la imagen inferior los campos que debemos rellenar unos obligatorios y otros opcionales pero recomendables a la hora de poder listarlos. Los dos mas importantes son:

- **Name:** representa el nombre único asociado al recurso que queremos proteger y **será utilizado posteriormente** cuando hagamos pruebas contra los mismos
- **Display Name:** breve descripción del recurso para poder seleccionarlo o listarlo fácilmente.
- **Type:** URD o ID único asociado al recurso
- **URI:** url asociado al recurso, representa el endpoint implementado en nuestra API
- **Scopes:** representan los scopes que este recurso implementa y como hemos dicho en el enunciado los dos recursos: producto y cliente implementan un CRUD, por lo que añadiremos estos scopes a los dos recursos creados: client-resource y product-resource.

Client-resource

Name: client-resource

Display name: Client Resource

Owner: poc-backend

Type: urn:poc-frontend:resources:client

URI: /clients/\*

Scopes: [x view] [x edit] [x delete] [x create]

Icon URI:

User-Managed Access Enabled: OFF

Key	Value	Actions
		Add

Save Cancel

Aquí podemos ver la lista de todos los recursos que hemos creado para poder gestionar todos los recurso del enunciado.

Poc-backend

Settings Credentials Keys Roles Client Scopes Mappers Scope Authorization Revocation Sessions Offline Access

Service Account Roles

Settings Resources Authorization Scopes Policies Permissions Evaluate Export Settings

Filter: Name Type URI Owner Scope

Name	Type	URIS	Owner
> client-resource	urn:poc-frontend:resources:client	/clients/*	poc-backend
> product-resource	urn:poc-frontend:resources:product	/products/*	poc-backend

### STEP 03: Crear las políticas de acceso y permisos asociados a ellas

Una vez ya hemos creado los recursos que queremos proteger junto a los scopes que cada uno de ellos implementa, vamos a pasar a esta tercera fase en donde ya vamos a configurar las políticas y finalmente los **permisos que aglutinan los anteriores objetos creados: scopes, resource y policies.**

En este diagrama se puede ver los pasos a seguir en esta tercera fase antes descrita



Primero vamos a crear las políticas de acceso a los recursos, que como hemos explicado anteriormente representan las condiciones que se deben de cumplir para acceder a los recursos. En nuestro caso vamos a crear **políticas basadas en rol** (RBAC) pues estos objetos son los que vamos a utilizar para segregar a nuestros usuarios dentro de la aplicación a nivel de autorización, pero tener presente que existen muchas otros tipos de políticas que podemos configurar dentro de la **pestaña de Policies**.

Una tabla que podría definir estas políticas de tipo role:

Policy	Role
Admin Policy	admin
Operator Policy	operator
User Policy	user

Podemos ver como hemos creado una política de acceso a los recursos para cada uno de los roles que hemos definido a la hora de crear nuestro realm, en este caso, roles de tipo realm globales a todos los clientes de nuestro realm.

Para crear una policy de este tipo debemos de seleccionar el tipo Role y rellenar:

- **Name:** nombre único de nuestra policy
- **Description:** breve descripción de la policy, para poder filtrarla en una lista.
- **Roles:** conjunto de roles al que se le debe aplicarse

▼

Realm Settings

Clients

Client Scopes

Roles

Identity Providers

User Federation

Authentication

Groups

Users

Sessions

Events

Clients > poc-backend > Authorization > Policies > Roles > Admin Policy

Admin Policy

Name \* ⓘ

Admin Policy

Description ⓘ

Policy that grants access only for users within admin role

Realm Roles \* ⓘ

Select a role...

Name	Required	Actions
admin	<input type="checkbox"/>	<div>Remove</div>

Clients ⓘ

Select One.....

Client Roles \* ⓘ

Select a role...

Logic ⓘ

Positive

Save

Cancel

Aqui podemos ver las políticas que hemos creado para los dos recursos

Poc

Configure

Realm Settings

Clients

Client Scopes

Roles

Identity Providers

User Federation

Authentication

Manage

Groups

Users

Sessions

Events

Import

Clients > poc-backend > Authorization > Policies

Poc-backend

Settings

Credentials

Keys

Roles

Client Scopes ⓘ

Mappers ⓘ

Scope ⓘ

Authorization

Revocation

Sessions ⓘ

Offline Access ⓘ

Clustering

Installation ⓘ

Settings

Resources

Authorization Scopes

Policies

Permissions

Evaluate

Export Settings

Filter: Name ⓘ Resource ⓘ Scope ⓘ All types ⓘ

Create Policy... ⓘ

Name	Description	Type	Actions
Admin Policy	Policy that grants access only for users within admin role	role	<div>Delete</div>
Operator Policy	Policy that grants access only for users within operator role	role	<div>Delete</div>
User Policy	Policy that grants access only for users within user role	role	<div>Delete</div>

Ahora que ya tenemos definido todos los objetos necesarios para crear los permisos: **scopes**, **resources** y **permisos**, podemos empezar a crear los permisos con la granulometría necesaria cumpliendo las condiciones de la **tabla de permisos** antes descrita:

Vamos a visualizar algunos ejemplos que implementan esta tabla de permisos.

Por ejemplo el permisos de acceso al recurso de clientes para la edición (PUT). Poemos ver como los parámetros a rellenar son:

- **Name:** nombre único del permiso
- **Descripción:** Descripción del permiso para poder filtrarlo en una lista
- **Resource:** sobre que recurso se aplica el permiso
- **Scope:** para que scopes de todos los que acepta el Resource se aplicará el permiso
- **Policy:** cuales son las políticas que van a controlar que se acepte el permiso.
- **Strategy Decision:** que estrategia se va a aplicar a la hora de ejecutar la policy. Escojaremos la de **Affirmative**, que indica que al menos una policy se debe de cumplir para aceptar el permiso.

Clients > poc-backend > Authorization > Permissions > client-edit

### Client-edit

**Name \*** client-edit

**Description** A permission that applies to the client resource type for edit scope

**Resource** client-resource

**Scopes \*** edit

**Apply Policy**

Select existing policy... Create Policy

Name	Description	Actions
Operator Policy	Policy that grants access only for users within operator role	Remove
Admin Policy	Policy that grants access only for users within admin role	Remove

**Decision Strategy** Affirmative

Save Cancel

Finalmente podemos ver todos los permisos que hemos configurado para implementar la tabla de permisos antes descrita.

Clients > poc-backend > Authorization > Permissions

### Poc-backend

Settings Credentials Keys Roles Client Scopes Mappers Scope Authorization Revocation Sessions Offline Access Clustering Installation

Service Account Roles

Settings Resources Authorization Scopes Policies Permissions Evaluate Export Settings

Filter: Name	Resource	Scope	All types	Create Permission...
Name	Description	Type	Actions	
client-create	A permission that applies to the client resource type for create scope	scope	Delete	
client-delete	A permission that applies to the client resource type for delete scope	scope	Delete	
client-edit	A permission that applies to the client resource type for edit scope	scope	Delete	
client-view	A permission that applies to the client resource type for view scope	scope	Delete	
product-create	A permission that applies to the product resource type for create scope	scope	Delete	
product-delete	A permission that applies to the product resource type for delete scope	scope	Delete	
product-edit	A permission that applies to the product resource type for edit scope	scope	Delete	
product-view	A permission that applies to the product resource type for view scope	scope	Delete	

## Evaluación

La herramienta de Autorización de Keycloak cuenta con una pestaña llamada **Evaluate**, desde donde podemos probar si la configuración que hemos creado cumple con la tabla de permisos antes descrita.

En ella podemos seleccionar muchas combinaciones para poder probar mucha casuística.

Por ejemplo vamos a probar como **un usuario con role Operador puede crear Productos pero no Clientes**.

Empezamos chequeando la **creacion de productos** para un **usuario de tipo operdor**

[Previous Evaluation](#)

## Identity Information ?

Client ?

poc-backend

User \* ?

operator

x

Roles ?

Any role...

## > Contextual Information ?

## Permissions ?

Apply to Resource Type ?

OFF

Resources \* ?

Select a resource...

Scopes ?

Any scope...

Add

Resource	Scopes	Actions
product-resource	create	<div>Delete</div>

Evaluate

Reset

El **resultado** de la evaluación es el esperado, que **SI puede crear productos**:

[Settings](#) [Resources](#) [Authorization](#) [Scopes](#) [Policies](#) [Permissions](#) [Evaluate](#) [Export Settings](#)

[Back](#) | [Re-Evaluate](#) | [Show Authorization Data](#)

## ∨ product-resource with scopes [create]

Result ?

PERMIT

Scopes ?

• create

- Policies
- **product-create** decision was **PERMIT** by **AFFIRMATIVE** decision. Granted Scopes: **create**.
    - **Operator Policy** voted to **PERMIT**.
    - **Admin Policy** voted to **DENY**.
  - **product-view** decision was **PERMIT** by **AFFIRMATIVE** decision. Granted Scopes: **view**.
    - **User Policy** voted to **DENY**.
    - **Operator Policy** voted to **PERMIT**.
    - **Admin Policy** voted to **DENY**.
  - **product-delete** decision was **PERMIT** by **AFFIRMATIVE** decision. Granted Scopes: **delete**.
    - **Operator Policy** voted to **PERMIT**.
    - **Admin Policy** voted to **DENY**.
  - **product-edit** decision was **PERMIT** by **AFFIRMATIVE** decision. Granted Scopes: **edit**.



- **Operator Policy** voted to **PERMIT**.
- **Admin Policy** voted to **DENY**.

Ahora vamos chequear el otro recurso la **creación de clientes** para el mismo **tipo de usuario con role operador**

Settings Resources Authorization Scopes Policies Permissions Evaluate Export Settings

[Previous Evaluation](#)

Identity Information ⓘ

Client ⓘ

poc-backend

User \* ⓘ

operator

x

Roles ⓘ

Any role...

> Contextual Information ⓘ

Permissions ⓘ

Apply to Resource Type ⓘ

OFF

Resources \* ⓘ

Select a resource...

Scopes ⓘ

Any scope...

Add

Resource	Scopes	Actions
client-resource	create	<div>Delete</div>

Evaluate

Reset

El resultado es el esperado que es que **NO puede crear Clientes**:

Settings Resources Authorization Scopes Policies Permissions Evaluate Export Settings

[Back](#) | [Re-Evaluate](#) | [Show Authorization Data](#)

✓ client-resource with scopes [create]

Result ⓘ

DENY

Scopes ⓘ

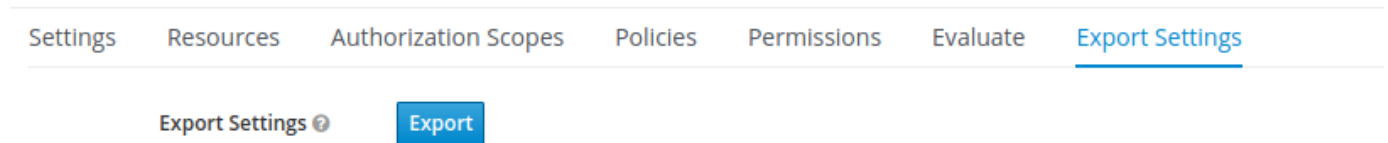
No scopes available.

Policies

- **client-view** decision was **PERMIT** by **AFFIRMATIVE** decision. Granted Scopes: **view**.
  - **User Policy** voted to **DENY**.
  - **Operator Policy** voted to **PERMIT**.
  - **Admin Policy** voted to **DENY**.
- **client-create** decision was **DENY** by **AFFIRMATIVE** decision. Denied Scopes: **create**.

- **Admin Policy** voted to **DENY**.
- **client-edit** decision was **PERMIT** by **AFFIRMATIVE** decision. Granted Scopes: **edit**.
  - **Operator Policy** voted to **PERMIT**.
  - **Admin Policy** voted to **DENY**.
- **client-delete** decision was **DENY** by **AFFIRMATIVE** decision. Denied Scopes: **delete**.
  - **Admin Policy** voted to **DENY**.

Estas evaluaciones las podemos ejecutar para todos los permisos que hemos creado fácilmente con esta herramienta ofrecida por Keycloak. También podemos exportar toda esta configuración de permisos desde la última pestaña



## Integración sistema externos: Unit Tests

Vamos a ver algunas unidades de test que prueban igualmente que lo que hemos validado desde la herramienta interna de Keycloak sigue funcionando cuando ejecutamos los endpoints ofrecidos por Keycloak a tal efecto.

Para hacer estas unidades de test vamos a utilizar la herramienta de Postman, por su flexibilidad a la hora de diseñar y testear todo tipo de APIs. En concreto vamos a utilizar 3 endpoints que voy a describir a continuación:

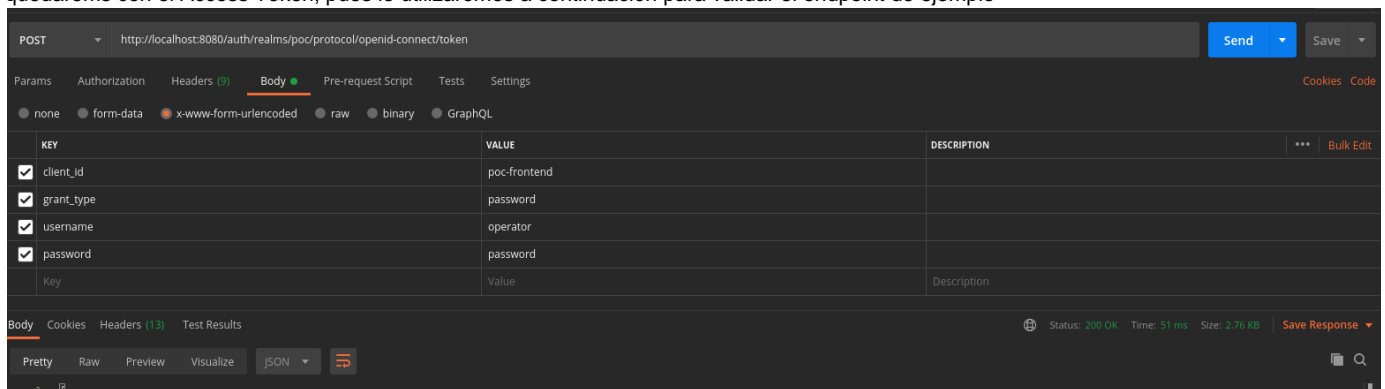
### 1. Request Token:

Este endpoint sirve para recuperar Access Tokens a partir de la autenticación del mismo dado su username y password correctos.

La estructura de la petición y sus atributos es la siguiente:

- **URI:** <http://localhost:8080/auth/realms/poc/protocol/openid-connect/token>
- **Header:** Content-Type: application/x-www-form-urlencoded
- **Body:**
  - **client\_id:** poc-frontend (este es el cliente público accesible por el frontend para autenticar usuarios)
  - **grant\_type:** password
  - **username:** operator
  - **password:** password

Aquí podemos ver en esta captura la petición con todos el cuerpo de la llamada así como el resultado de la misma, principalmente nos quedaremos con el Access Token, pues lo utilizaremos a continuación para validar el endpoint de ejemplo



POST

http://localhost:8080/auth/realms/poc/protocol/openid-connect/token

Send

Save

Params

Authorization

Headers (10)

Body

Proxypoint Script

Tabs

Settings

Cookies

Code

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> audience	poc-backend	
<input checked="" type="checkbox"/> grant_type	urn:ietf:params:oauth:grant-type:uma-ticket	
<input checked="" type="checkbox"/> permission	client-resource#create	
<input checked="" type="checkbox"/> response_mode	decision	
Key	Value	Description

Body	Cookies	Headers (1)	Test Results	Status: 403 Forbidden	Time: 8 ms	Size: 433 B	Save Response
------	---------	-------------	--------------	-----------------------	------------	-------------	---------------

```

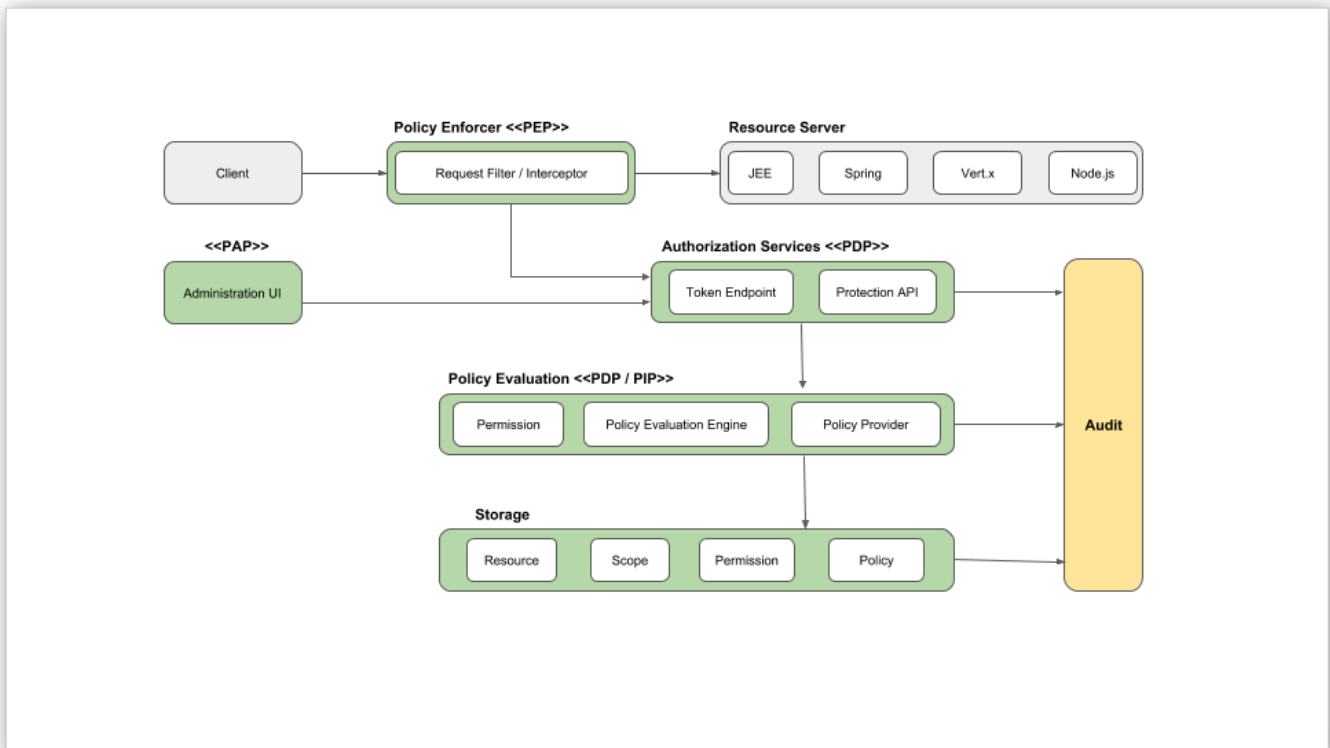
1 {
2   "error": "access_denied",
3   "error_description": "not_authorized"
4 }

```


### 3. Get Party Token (RPT)

Para obtener la lista de permisos de Keycloak debemos de enviar una **request para obtener un Party Token (RPT)**. Como resultado Keycloak evaluará todas las políticas asociadas con los resources y scopes pedidos y enviará un Party Token con todos los permisos concedidos por el servidor. En el diagrama podemos ver todos los elementos que existen en un proceso de autorización centralizado como el que hemos estado hablando:


- **PEP:** es el patron de diseños que deberemos de implementar dentro de nuestro servidor de recursos en el lenguaje que sea para que poder comunicarnos con la implementación centralizada de Keycloak
- **PAP:** son las vistas ofrecidas por Keycloak desde el Admin console dedicadas a configurar las todas entidades involucradas en esta implementación: Resources, Scopes, Policy y Permission. Podremos nosotros implementarlo dentro de nuestra aplicación utilizando los módulos de Admin API y Authorization Services que suelen existir en los adaptadores de varios lenguajes.
- **PDP:** son los endpoints expuestos por Keycloak encargados de recuperación los Party tokens y de validar el acceso a los recursos de nuestro servidor controlados por Keycloak.
- **PIP:** es la implementación de Keycloak encargada de ejecutar las peticiones procedentes de PDP



Vamos a seguir probándolo utilizando el usuario operador.

JWT

DebuggerLibrariesIntroductionAsk

Crafted by auth0

```
ob3JpemF0aW9uIiwib3BlcmF0b3IiXX0sInJlc2
91cmNlX2FjY2VzcyI6eyJhY2NvdW50IjpbInJvb
GVzIjpbIm1hbmFnZS1hY2NvdW50IiwibWFuYXd1
LWFjY291bnQtbGlua3MiLCJ2aWV3LXByb2ZpbGU
iXX19LCJhdXRob3JpemF0aW9uIjpbInBlcm1pc3
Npb25zIjpbeyJzY29wZXMiOlsidmlldyIsImVka
XQiXSwicnNpZCI6IjJmNTg0GRjLTE0YTctNGM4
Ny1hNzI2LTd1ODh1NjFhMjAzZiIsInJzbmFtZSI
6ImNsaWVudC1yZXNvdXJzZSJ9LHsic2NvcGVzIj
pbInZpZXciLCJlZG10Iiw1Y3JlYXR1Iiw1ZGVsZ
XR1IiIsInJzaWQiOiIxZmU5YmI1NC00ZjlhLTQ1
Y2MtOTc1OC0zNzAzYjAyOTRlNWYiLCJyc25hbWU
iOiJwcm9kdWN0LXJlc291cmNlIn1dfSwic2NvcG
UiOiJlbWFPbCBwcm9maWx1Iiwic2lkIjo1ZWFKY
TUyMmYtYTE1Zi00ZDVhLThkYjMtYjgwYjYjVjMTVl
MmY4Iiw1ZG10Iiw1Y3JlYXR1Iiw1ZGVsZ
wcmVmZXJyZWRfdXN1cm5hbWUiOiJvcGVyYXRvci
J9.PGMVafnEs7Y7R860QxJBN1_wzcFPSnwohuTJ
UividGGkSgN0y70T6L48tSWk9fd17YCCIDdvDvs
L0UeQ2CQPLX5mWCL6bTiIoadYD9UeU1Z81sCru_
3iM0rviSeWTBASGJumk12pv9xVcamVdcJ2AG4Qy
Vjs0CL5KRmt2kB_CzeFf-
H5dz9D3U18A16aWGnbiQ8fW5cM98-
```

```
{
  "realm_access": {
    "roles": [
      "default-roles-oliverealms",
      "offline_access",
      "uma_authorization",
      "operator"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "authorization": {
    "permissions": [
      {
        "scopes": [
          "view",
          "edit"
        ],
        "rsid": "2f5888dc-14a7-4c87-a726-7e88e61a203f",
        "rsname": "client-resource"
      },
      {
        "scopes": [
          "view",
          "edit",
          "*****"
        ]
      }
    ]
  }
}
```

Qy99QeY9PmtXbFZqMui\_qe6vrvHPM3eiTrCVmbU  
TjN8f5UFSscAVwbw2423BkSi2PVCoicxC5U93lc  
Y-IxYcgzvKQhKBTap-  
vbREX4SoiuVHKdHttD8lnKJ-  
ecb\_0lpmhqA8EvbKr7Q

```
        "delete",
      ],
      "rsid": "1fe9bb54-4f9a-45cc-9758-3703b0294e5f",
      "rsname": "product-resource"
    }
  ],
},
```

Donde se puede ver toda la informacion ofrecida por el party token especialmente la relacionada con la autorización: roles y permisos que son los esperados dada la configuración realizada previamente desde Keycloak:

- **Role** de tipo **operator**
- Y los **siguientes permisos**:

Resource	Scopes
client-resource	view, edit
product-resource	view, edit, create, delete