FLOWX.AI

PLATFORM SETUP GUIDES

Contents

PLATFORM SETUP GUIDES / Overview

The setup guides in this section will provide information on how to install, configure, and use FLOWX.AI services.

Deploying microservices typically involves breaking down the application into smaller, modular components. Each microservice should be independently deployable, with all the necessary dependencies and configurations included.

Once the microservices have been created, they can be deployed using a container management system such as Docker or Kubernetes. These systems allow for the deployment of multiple microservices in a single environment.

Environment variables

Environment variables are variables that are set in the system environment and can be used by applications and services to store and access configuration information. Environment variables typically include settings such as paths to

directories, file locations, settings for the operating system and applications, and more.

Environment variables are used to store and access configuration information in a secure and efficient manner. Below you will find some examples of common/shared environment variables that need to be set for different services and components.

Authorization & access roles

An identity management platform is a software system that helps you manage authorization & access roles, including user accounts, passwords, access control, and authentication. Identity management platforms typically offer features such as user provisioning, identity federation, and single sign-on.

The following variables need to be set in order to connect to the identity management platform:

- `SECURITY_OAUTH2_BASE_SERVER_URL` - the base URL for the OAuth 2.0 Authorization Server, which is responsible for authentication and authorization for clients and users, it is used to authorize clients, as well as to issue and validate access tokens
- `SECURITY_OAUTH2_CLIENT_CLIENT_ID` - a unique identifier for a client application that is registered with the OAuth 2.0 Authorization Server, this is used to authenticate the client application when it attempts to access resources on behalf of a user
- `SECURITY_OAUTH2_CLIENT_CLIENT_SECRET` - secret key that is used to authenticate requests made by an authorization client

- `SECURITY_OAUTH2_REALM` - security configuration env var in the Spring Security OAuth2 framework, it is used to specify the realm name used when authenticating with OAuth2 providers

» Access Management

Datasource configuration

Datasource configuration is the process of configuring a data source, such as a database, file, or web service, so that an application can connect to it and use the data. This typically involves setting up the connection parameters, such as the host, port, username, and password.

In some cases, additional configuration settings may be required, such as specifying the type of data source (e.g. Oracle, MySQL, etc.) or setting up access control for data access.

Environment variables are more secure than hard-coding credentials in the application code and make it easier to update data source parameters without having to modify the application code.

CAUTION

Some microservices (**Admin** microservice, for example, connects to the same Postgres / Oracle database as the **Engine**).

INFO

> Depending on the data source type, various parameters may need to be configured. For example, if connecting to an Oracle database, the driver class name, and the database schema must be provided. For MongoDB, the URI is needed.

The following variables need to be set in order to set the datasource:

- `SPRING_DATASOURCE_URL` - environment variable used to configure a data source URL for a Spring application, it typically contains the JDBC driver name, the server name, port number, and database name
- `SPRING_DATASOURCE_USERNAME` - environment variable used to set the username for the database connection, this can be used to connect to a database instance
- `SPRING_DATASOURCE_PASSWORD` - environment variable used to store the password for the database connection, this can be used to secure access to the database and ensure that only authorized users have access to the data
- `SPRING_DATASOURCE_DRIVERCLASSNAME` (❗ only for Oracle DBs) - environment variable used to set the class name of the JDBC driver that the Spring DataSource will use to connect to the database
- `SPRING_JPA_PROPERTIES_HIBERNATE_DEFAULTSCHEMA` (❗ only for Oracle DBs) - environment variable used to overwrite the name of the database schema
- `SPRING_DATA_MONGODB_URI` (❗ only for MongoDB) - environment variable used to provide the connection string for a MongoDB database that is used

with, this connection string provides the host, port, database name, user credentials, and other configuration details for the MongoDB server

> **CAUTION**
>
> You will need to make sure that the user, password, connection link and db name are configured correctly, otherwise, you will receive errors at start time.

> **INFO**
>
> The datasource is configured automatically via a liquibase script inside the engine. All updates will include migration scripts.

Kafka

The following Kafka-related configurations can be set by using environment variables:

- `SPRING_KAFKA_BOOTSTRAP_SERVERS` - environment variable used to configure the list of brokers to which the kafka client will connect, this is a comma-separated list of host and port pairs that are the addresses of the Apache Kafka brokers in a Kafka cluster
- `SPRING_KAFKA_CONSUMER_GROUP_ID` - environment variable is used to set the consumer group ID for the Kafka consumer, it is used to identify which consumer group the consumer belongs to and allows the Kafka broker to manage which messages are consumed by each consumer in the group

> **INFO**
>
> `SPRING_KAFKA_CONSUMER_GROUP_ID` - might be different for the services that have the group id separated in topics, also thread numbers.

- `KAFKA_CONSUMER_THREADS` - environment variable used to control the number of threads that a Kafka consumer instance can use to consume messages from a cluster, it defines the number of threads that the consumer instance should use to poll for messages from the Kafka cluster
- `KAFKA_AUTH_EXCEPTION_RETRY_INTERVAL` - environment variable used to set the interval at which Kafka clients should retry authentication exceptions (the interval between retries after AuthorizationException is thrown by KafkaConsumer)
- `KAFKA_MESSAGE_MAX_BYTES` - this is the largest size of the message that can be received by the broker from a producer.

Each action available in the service corresponds to a Kafka event. A separate Kafka topic must be configured for each use case.

> **CAUTION**
>
> FLOWX.AI Engine is listening for messages on topics with names of a certain pattern, make sure to use correct outgoing topic names when configuring the services.

Redis configuration

Redis configuration involves setting up the connection parameters, such as the host, port, username, and password. In some cases, additional configuration settings may be required, such as specifying the type of data store or setting up access control for data access.

- `SPRING_REDIS_HOST` - environment variable used to configure the hostname or IP address of a Redis server when using Spring Data Redis
- `SPRING_REDIS_PASSWORD` - environment variable is used to store the password used to authenticate with a Redis server, it is used to secure access to the Redis server and should be kept confidential
- `REDIS_TTL` - environment variable is used to specify the maximum time-to-live (TTL) for a key in Redis, it is used to set a limit on how long a key can exist before it is automatically expired (Redis will delete the key after the specified TTL has expired)

Debugging

Advanced debugging features can be enabled. When this happens, snapshots of the process status will be taken after each action and can be later used for debugging purposes. This feature comes with an exponential increase in database usage, so we suggest having the flag set to true on debugging media and false production ones.

Logging

The following environment variables could be set in order to control log levels:

- `LOGGING_LEVEL_ROOT` - root spring boot microservice logs

- `LOGGING_LEVEL_APP` - controls the verbosity of the application's logs and how much information is recorded (app level logs)

Tracing via Jaeger

Tracing via Jaeger involves collecting timing data from the components in a distributed application. This allows you to better identify bottlenecks and latency issues.

The following FLOWX.AI services use Jaeger tracing:

1. **scheduler-core**
2. **customer-management-plugin**
3. **document-plugin**
4. **notification-plugin**
5. **process-engine**

Environment variables to be set for tracing:

- `APPLICATION_JAEGER_ENABLED` - environment variable used to enable or disable Jaeger tracing
- `APPLICATION_JAEGER_PREFIX` - environment variable used to change the name in the Jaeger dashboard

License model

A license model is a set of rules and regulations governing how software can be used, distributed, and modified. It also outlines the rights and responsibilities of the

software user and the software developer. Common license models include open source, freeware, shareware, and commercial software.

Most of the **third-party components used by FLOWX.AI** are under **Apache License 2.0** source code.

Third-party components

Third-party components are software components or libraries that are not part of FLOWX.AI but are instead created by another company or individual and used in a development project.

These components can range from databases and operating systems to user interface components and libraries that provide support for a specific feature or task.

Third party components are components such as libraries, frameworks, APIs, etc.

» Third-party components

PLATFORM SETUP GUIDES / FLOWX.AI Engine setup guide / Advancing Controller setup guide

This guide provides step-by-step instructions to help you configure and deploy the Advancing Controller effectively.

Infrastructure prerequisites

Advancing controller requires the following components to be set up before it can be started:

- **FLOWX.AI Engine deployment** - the Advancing Controller is dependent on the FLOWX.AI Engine and must be deployed in the same environment, refer to the FLOWX.AI Engine setup guide for more information on how to set up the Engine
- **DB instance** - the Advancing Controller uses a PostgreSQL or OracleDB as database instance

Dependencies

- Database
- Datasource
- FLOWX.AI Engine

Database configuration

Postgres

A basic Postgres configuration for Advancing:

```yaml
postgresql:
  enabled: true
  postgresqlUsername: "postgres"
  postgresqlPassword: ""
  postgresqlDatabase: "advancing"
  existingSecret: "postgresql-generic"
  postgresqlMaxConnections: 200
  persistence:
    enabled: true
    storageClass: standard-rwo
    size: 20Gi
  resources:
    limits:
      cpu: 1000m
      memory: 1024Mi
    requests:
      memory: 256Mi
      cpu: 100m
  metrics:
    enabled: true
    serviceMonitor:
      enabled: false
    prometheusRule:
      enabled: false
  primary:
    nodeSelector:
      preemptible: "false"
```

CAUTION

If the parallel advancing configuration already exists, resetting the 'advancing' database must be done by executing the SQL command `DROP DATABASE`

> `advancing;`. Once the database has been dropped, the Liquibase script will automatically re-enable it.

Configuration

The following configuration details need to be added using environment variables:

Advancing controller uses a PostgreSQL or an Oracle database as a dependency.

- the user, password, connection link, and database name need to be configured correctly, if these details are not configured correctly, errors will occur at startup
- the datasource is configured automatically via a Liquibase script inside the engine. All updates will include migration scripts.

Configuring datasource

The following configuration details need to be added using environment variables:

- `SPRING_DATASOURCE_URL` - environment variable used to configure a data source URL for a Spring application, it typically contains the JDBC driver name, the server name, port number, and database name
- `SPRING_DATASOURCE_USERNAME` - environment variable used to set the username for the database connection, this can be used to connect to a database instance
- `SPRING_DATASOURCE_PASSWORD` - environment variable used to store the password for the database connection, this can be used to secure access to

the database and ensure that only authorized users have access to the data

- `SPRING_JPA_DATABASE` - relevant because it is used to specify the type of database that the Spring application should connect to (accepted values: `oracle` or `postgresql`)
- `SPRING_JPA_PROPERTIES_HIBERNATE_DEFAULTSCHEMA` (❗ only for Oracle DBs) - dpecifies the default schema to use for the database (default value: `public`)

You will need to make sure that the user, password, connection link and db name are configured correctly, otherwise, you will receive errors at start time.

> **CAUTION**
>
> It's important to keep in mind that the Advancing Controller is tightly integrated with the FLOWX.AI Engine. Therefore, it is important to ensure that both the Engine and the Advancing Controller are configured correctly and are in sync.

Was this page helpful?

PLATFORM SETUP GUIDES / FLOWX.AI Engine setup guide / Configuring access rights for Engine

Granular access rights can be configured for restricting access to the Engine component.

Two different access authorizations are provided, each with specified access scopes:

1. **Manage-processes** - for configuring access for running test processes

Available scopes:

- **edit** - users are able to start processes for testing and to test action rules

2. **Manage-instances** - for configuring access for manipulating process instances

Available scopes:

- **read** - users can view the list of process instances
- **admin** - users are able to retry an action on a process instance token

The Engine service is preconfigured with the following default users roles for each of the access scopes mentioned above:

- **manage-processes**
    - edit:
        - ROLE_ADMIN_MANAGE_PROCESS_EDIT
        - ROLE_ADMIN_MANAGE_PROCESS_ADMIN
    - admin:

        - ROLE_ADMIN_MANAGE_PROCESS_ADMIN

- **manage-instances**
    - read:
        - ROLE_ENGINE_MANAGE_INSTANCE_READ
        - ROLE_ENGINE_MANAGE_INSTANCE_ADMIN
    - admin:
        - ROLE_ENGINE_MANAGE_INSTANCE_ADMIN

> **DANGER**
> These roles need to be defined in the chosen identity provider solution.

In case other custom roles are needed, you can configure them using environment variables. More than one role can be set for each access scope.

To configure access for each of the roles above, adapt the following input:

`SECURITY_ACCESSAUTHORIZATIONS_AUTHORIZATIONNAME_SCOPES_SCOPENAME_ROLESALLOWED:NEEDED_ROLE_NAMES`

Possible values for AUTHORIZATIONNAME: MANAGEPROCESSES, MANAGEINSTANCES.

Possible values for SCOPENAME: read, edit, admin.

For example, if you need to configure role access for read, insert this:

```
SECURITY_ACCESSAUTHORIZATIONS_MANAGEINSTANCES_SCOPES_READ_ROLES
ROLE_NAME_TEST
```

Was this page helpful?

PLATFORM SETUP GUIDES / FLOWX.AI Engine setup guide / Configuring access roles for processes

Access to a process definition

Setting up user role-based access on process definitions is done by configuring swimlanes on the process definition.

» Swimlanes

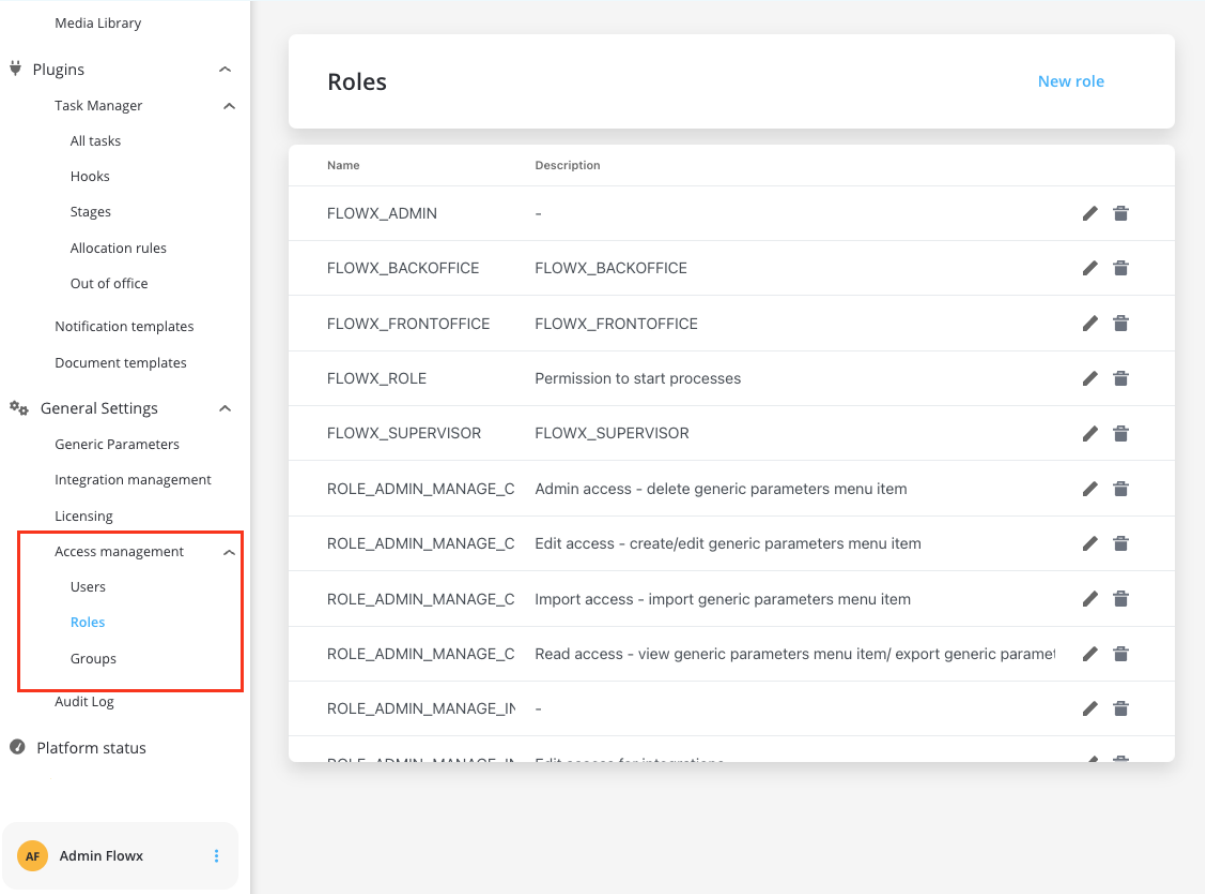
By default, all process nodes belong to the same swimlane. If more swimlanes are needed, they can be edited in the process definition settings panel.

Swimlane role settings apply to the whole process, the process nodes or the actions to be performed on the nodes.

First, the desired user roles need to be configured in the identity provider solution and users must be assigned the correct roles.

INFO

You can use the **Access management** tab under **General Settings** to administrate all the roles.



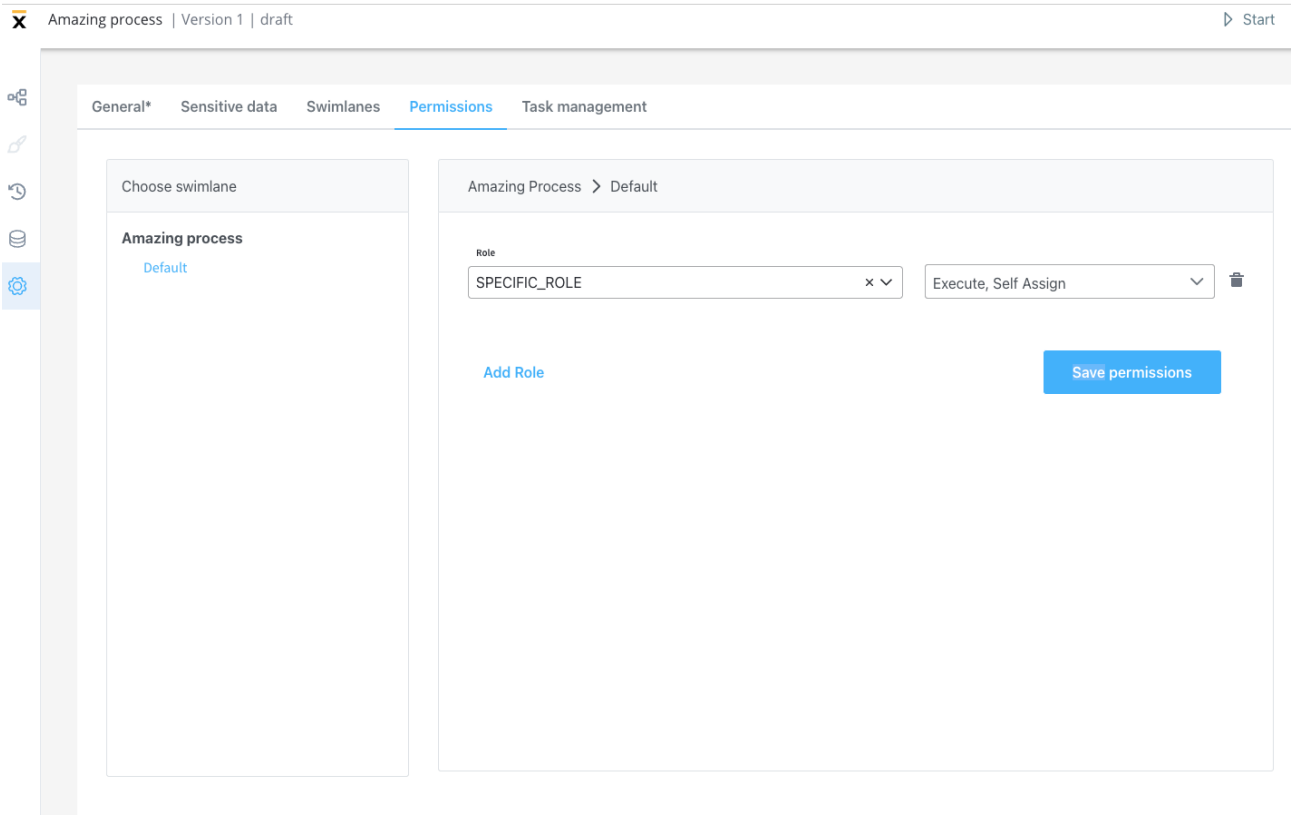
DANGER

To be able to access the roles defined in the identity provider solution, a **service account** with appropriate permissions needs to be added in the

identity provider. And the details of that service account **need to be set up in the platform configuration**.

The defined roles will then be available to be used in the process definition settings (**Permissions** tab) panel for configuring swimlane access.

A **Default** swimlane comes with two default permissions assigned based on a specific role.

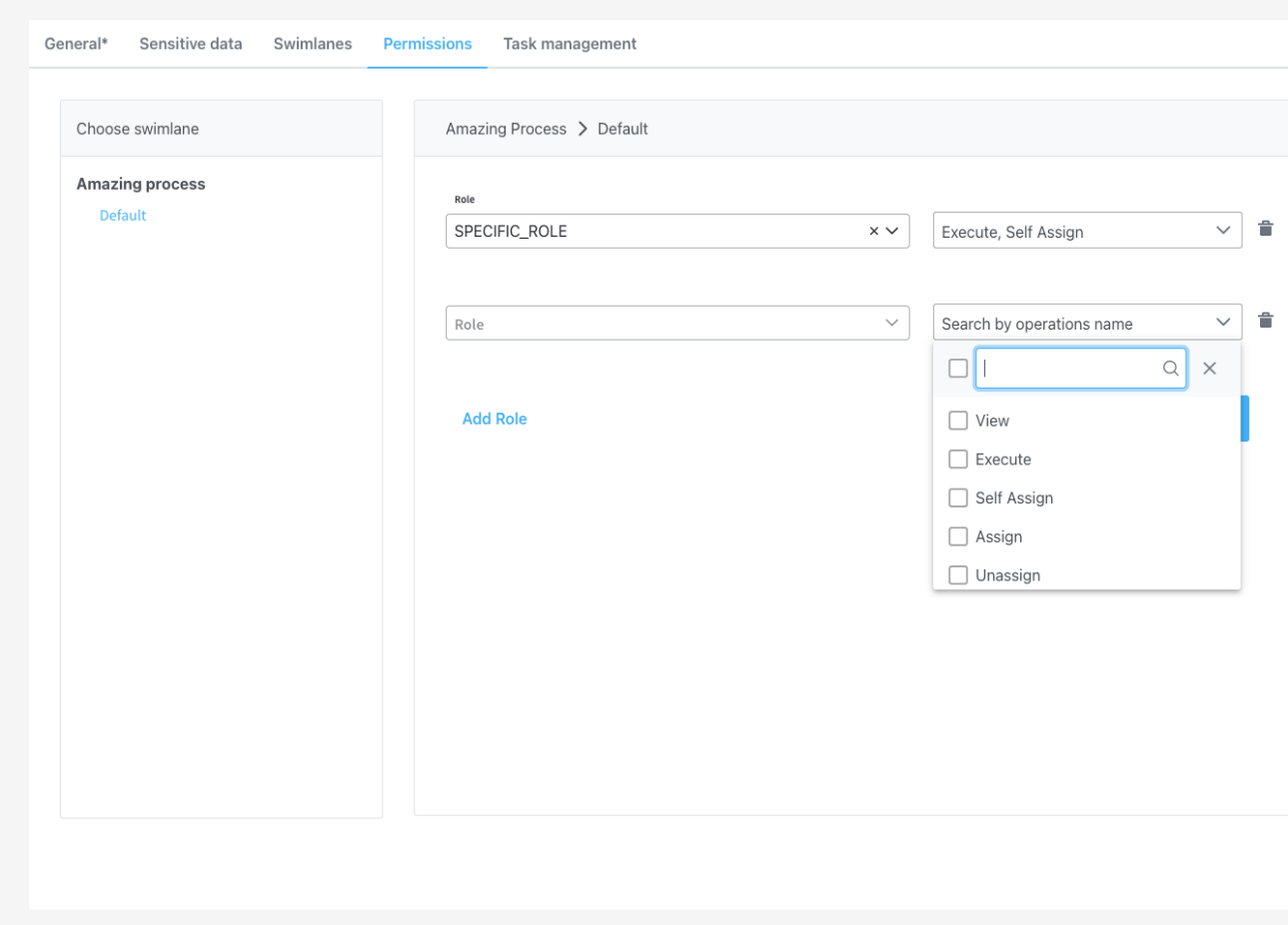


- **execute** - the user will be able to start process instances and run actions on them
- **self-assign** - the user can assign a process instance to them and start working on it

> **DANGER**
>
> This is valid for **> 2.11.0** FLOWX.AI platform release.

Other **Permissions** can be added manually, depending on the needs of the user. Some permissions are needed to be configured so you can use features inside Task Management plugin. Specific roles need to be assigned separately on a few available process operations. These are:

- **view** - the user will be able to view process instance data
- **assign** - user can assign tasks to other users (this operation is only accessible through the **Task management** plugin)
- **unassign** - user can unassign tasks from other users (this operation is only accessible through the **Task management** plugin)
- **hold** - user can mark the process instance as on hold (this operation is only accessible through the **Task management** plugin)
- **unhold** - user can mark the process instance as not on hold (this operation is only accessible through the **Task management** plugin)



DANGER

< 2.11.0 platform release - if no role is configured on an operation, no restrictions will be applied.

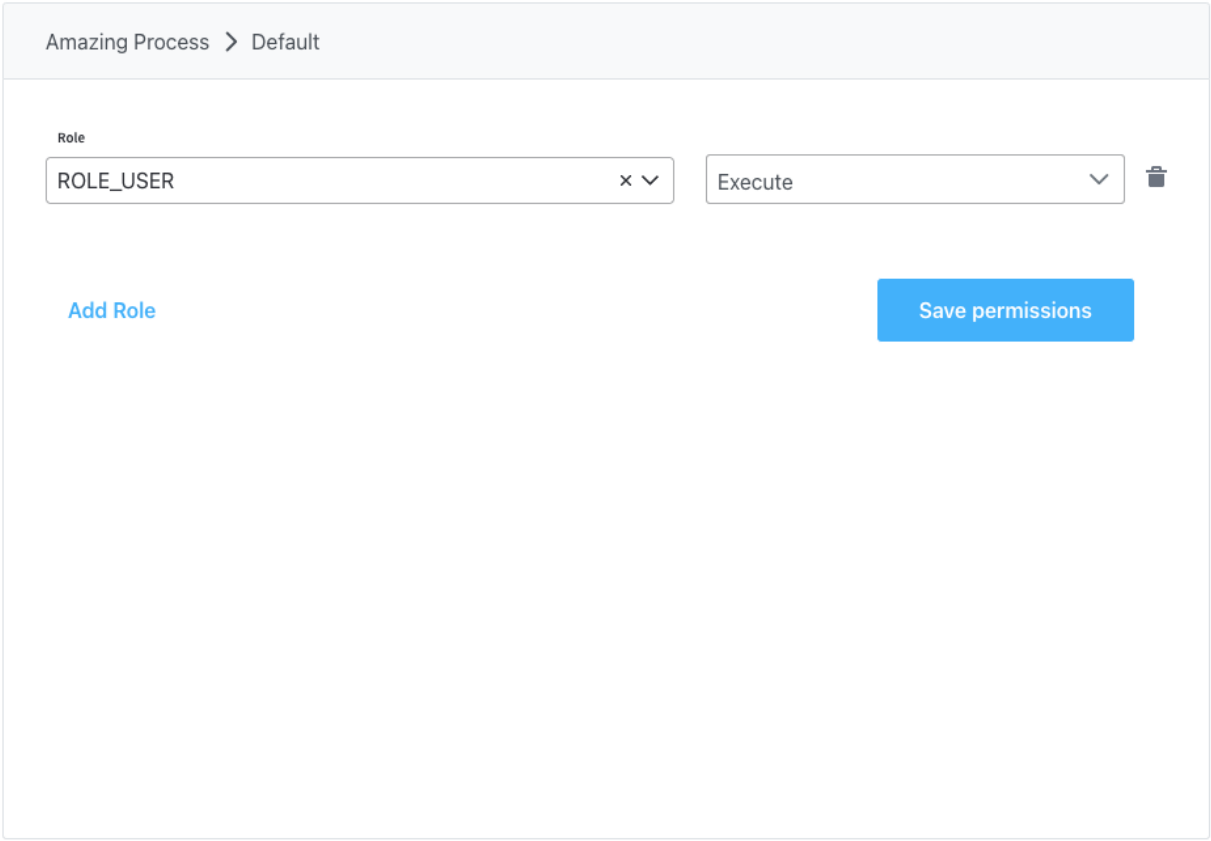
Configuration examples

CAUTION

Valid for < 2.11.0 release version.

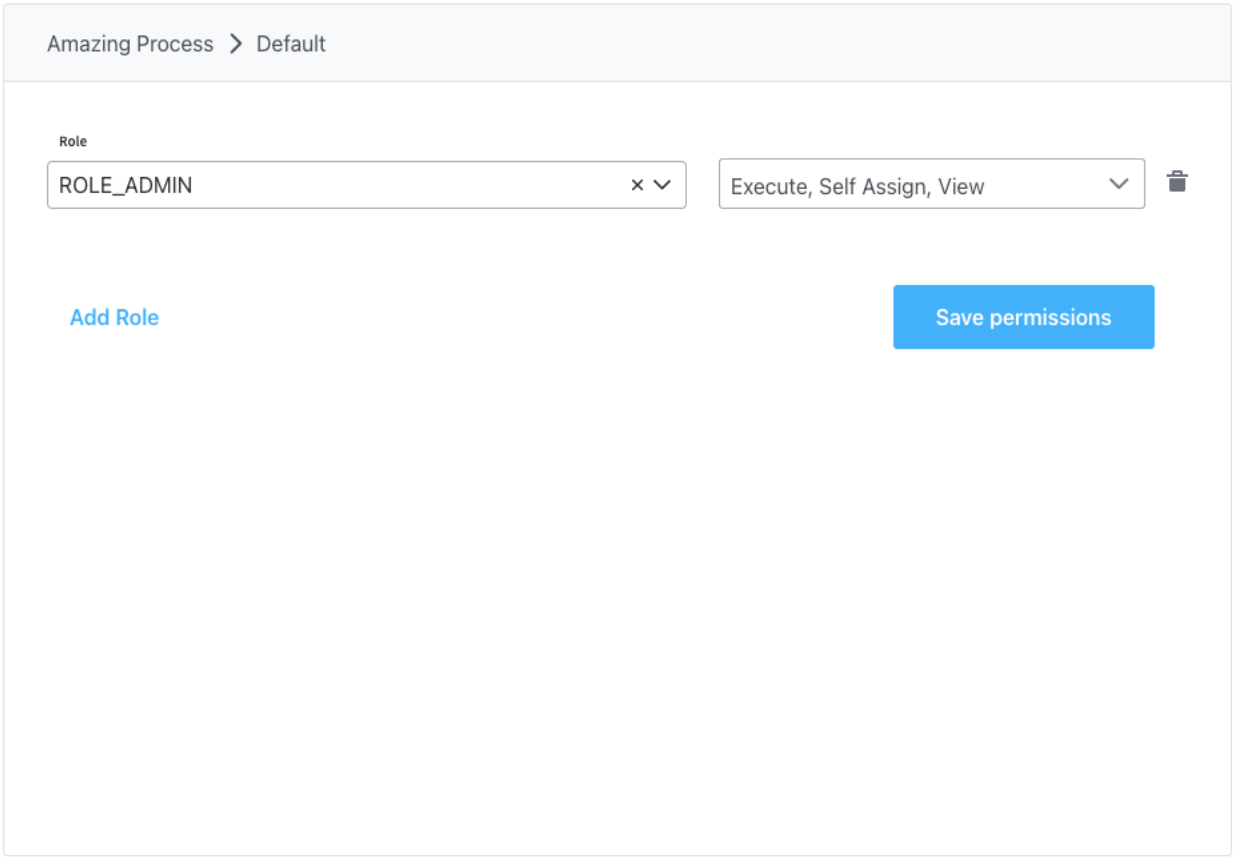
Regular user

Below you can find an example of configuration of roles for a regular user:



Admin

Below you can find an example of configuration of roles for an admin user:



CAUTION

! Starting with **2.11.0** release, specific roles are needed, otherwise, restrictions will be applied.

After setting up your preferred identity provider solution, you will need to add the desired access roles in the application configuration for the FLOWX Engine (using environment variables):

» Authorization & access roles

Restricting process instance access based on business filters

» Business filters

Before they can be used in the process definition the business filter attributes need to be set in the identity management platform. They have to be configured as a list of filters and should be made available on the authorization token. Application users will also have to be assigned this value.

Viewing processes instances

Active process instances and their related data can be viewed from the FLOWX Designer. A user needs to be assigned to a specific role in the identity provider solution to be able to view this information.

By default, this role is named `FLOWX_ROLE`, but its name can be changed from the application configuration of the Engine by setting the following environment variable:

`FLOWX_PROCESS_DEFAULTROLES`

When viewing process instance-related data, it can be configured whether to hide specific sensitive user data. This can be configured using the `FLOWX_DATA_ANONYMIZATION` environment variable.

Access to REST API

To restrict API calls by user role, you will need to add the user roles in the application config:

```
security:
  pathAuthorizations:
    -
      path: "/api/**"
      rolesAllowed: "ANY_AUTHENTICATED_USER" or
"USER_ROLE_FROM_IDENTITY_PROVIDER"
```

Was this page helpful?

PLATFORM SETUP GUIDES / FLOWX.AI Engine setup guide / Process instance indexing / Configuration guidelines

The configuration of Elasticsearch for process instances indexing depends on various factors related to the application load, the number of process instances, parallel requests, and indexed keys per process. Although the best approach to sizing and configuring Elasticsearch is through testing and monitoring under load, here are some guidelines to help you get started:

Indexing strategy

When deleting data in Elasticsearch, it's recommended to delete entire indices instead of individual documents. Creating multiple smaller indices provides the flexibility to delete entire indices of old data that are no longer needed.

- Advantages of Multiple Small Indices:
  - Fast indexing process.
  - Flexibility in cleaning up old data.
- Potential Drawbacks:
  - Hitting the maximum number of shards per node, resulting in exceptions when creating new indices.
  - Increased search response time and memory footprint.

Alternatively, you can create fewer indices that span longer periods of time, such as one index per year. This approach offers small search response times but may result in longer indexing times and difficulty in cleaning up and recovering data in case of failure.

» What is indexing?

Shard and replica configuration

The solution includes an index template that gets created with the settings from the process-engine app (name, shards, replicas) when running the app for the first time. This template controls the settings and mapping of all newly created indices.

» What is sharding?

» Index template

Once an index is created, you cannot update its number of shards and replicas. However, you can update the settings from the index template at runtime in Elasticsearch, and new indices will be created with the updated settings. Note that the mapping should not be altered as it is required by the application.

Recommendations for resource management

To manage functional indexing operations and resources efficiently, consider the following recommendations:

- Sizing indexes upon creation
- Balancing
- Delete unneeded indices
- Reindex large indices
- Force merge indices
- Shrink indices
- Combine indices

Sizing indexes upon creation

Recommendations:

- Start with monthly indexes that have 2 shards and 1 replica. This setup is typically sufficient for handling up to 200k process instances per day; ensures a parallel indexing in two main shards and has also 1 replica per each main

shard (4 shards in total). This would create 48 shards per year in the elastic search nodes; A lot less than the default 1000 shards, so you will have enough space for other indexes as well.
    - If you observe that the indexing gets really, really slow, then you should look at the physical resources / shard size and start adapting the config.
    - If you observe that indexing one monthly index gets massive and affects the performance, then think about switching to weekly indices.
    - If you have huge spikes of parallel indexing load (even though that depends on the Kafka connect cluster configuration), then think about adding more main shards.
- Consider having at least one replica for high availability. However, keep in mind that the number of replicas is applied to each shard, so creating many replicas may lead to increased resource usage.
- Monitor the number of shards created and estimate when you might reach the maximum shards per node, taking into account the number of nodes in your cluster.

Balancing

When configuring index settings, consider the number of nodes in your cluster. The total number of shards (calculated by the formula: primary_shards_number * (replicas_number +1)) for an index should be directly proportional to the number of nodes. This helps Elasticsearch distribute the load evenly across nodes and avoid overloading a single node. Avoid adding shards and replicas unnecessarily.

Delete unneeded indices

Deleting unnecessary indices reduces memory footprint, the number of used shards, and search time.

Reindex large indices

If you have large indices, consider reindexing them. Process instance indexing involves multiple updates on an initially indexed process instance, resulting in multiple versions of the same document in the index. Reindexing creates a new index with only the latest version, reducing storage size, memory footprint, and search response time.

Force merge indices

If there are indices with no write operations performed anymore, perform force merge to reduce the number of segments in the index. This operation reduces memory footprint and response time. Only perform force merge during off-peak hours when the index is no longer used for writing.

Shrink indices

If you have indices with many shards, consider shrinking them using the shrink operation. This reindexes the data into an index with fewer shards. Perform this operation during off-peak hours.

Combine indices

If there are indices with no write operations performed anymore (e.g., process_instance indices older than 6 months), combine these indices into a larger one and delete the smaller ones. Use the reindexing operation during off-peak hours. Ensure that write operations are no longer needed from the FLOWX platform for these indices.

Was this page helpful?

PLATFORM SETUP GUIDES / FLOWX.AI Engine setup guide / Old access roles

> **CAUTION**
>
> Deprecated since platform version 1.16.0

Old access roles

Access to a process definition

You can restrict access to process definitions by user roles. This can be done by setting the desired operation permissions on a process definition.

Start by adding the needed roles in the database. These need to match the roles configured in the identity provider solution. Each role can have one or more permissions defined on it. Permissions can be applied to all users or only to the owner of the specific resource (for example the person that started the process instance).

After saving a new process definition, you can also save specific user roles for it to restrict user access. Access rights can be defined on the following operations that can be performed on a process definition:

- starting a new instance of the process definition
- viewing the instance of that process definition

Here's an example of setting operation permissions for a process definition:

```
{
    "START": ["PROCESS_START"],
    "VIEW": ["PROCESS_VIEW", "PROCESS_VIEW_ALL"]
}
```

where `START` and `VIEW` are the possible operations to be performed on the definitions and `PROCESS_START`, `PROCESS_VIEW`, `PROCESS_VIEW_ALL` are permissions stored in the database.

Access to actions from process definitions

Operation permissions can also be set on specific nodes in order to restrict the access to the actions defined on that node. This can be done similarly to setting operation permissions on process definitions. The operation name to be used for nodes is `NODE_RUN`.

As nodes also hold the definitions for the user interface, deciding which user role can see a certain UI template can also be done by using node permissions. The templates linked to a node can only be viewed by a user that has the `NODE_RUN` permission on that node, if the access on that node is restricted.

Access to a process definition

You can restrict access to process definitions by user roles. This can be done by setting the desired operation permissions on a process definition.

Start by adding the needed roles in the database. These need to match the roles configured in the identity provider solution. Each role can have one or more

permissions defined on it. Permissions can be applied to all users or only to the owner of the specific resource (for example the person that started the process instance).

After saving a new process definition, you can also save specific user roles for it to restrict user access.

Access rights can be defined on the following operations that can be performed on a process definition:

- starting a new instance of the process definition
- viewing the instance of that process definition

Here's an example of setting operation permissions for a process definition:

```
{
    "START": ["PROCESS_START"],
    "VIEW": ["PROCESS_VIEW", "PROCESS_VIEW_ALL"]
}
```

where `START` and `VIEW` are the possible operations to be performed on the definitions and `PROCESS_START`, `PROCESS_VIEW`, `PROCESS_VIEW_ALL` are permissions stored in the database.

Access to actions from process definitions

Operation permissions can also be set on specific nodes in order to restrict the access to the actions defined on that node. This can be done similarly to setting operation permissions on process definitions. The operation name to be used for nodes is `NODE_RUN`.

As nodes also hold the definitions for the user interface, deciding which user role can see a certain UI template can also be done by using node permissions. The templates linked to a node can only be viewed by a user that has the `NODE_RUN` permission on that node, if the access on that node is restricted.

Was this page helpful?

PLATFORM SETUP GUIDES / Access management / Configuring an IAM solution

Recommended Keycloak setup

To configure a minimal required Keycloak setup, follow these steps:

- Create a new realm
  - Define available roles and realm-level roles assigned to new users.
- Create/import user roles and groups
- Create new users
- Add clients
  - Configure access type, valid redirect URIs, and enable necessary flows.
- Add role mappers
- Add service accounts
  - Set up **admin**, **task management**, and **process engine** service accounts.

> **INFO**
>
> Recommended keycloak version: **18.0.x**

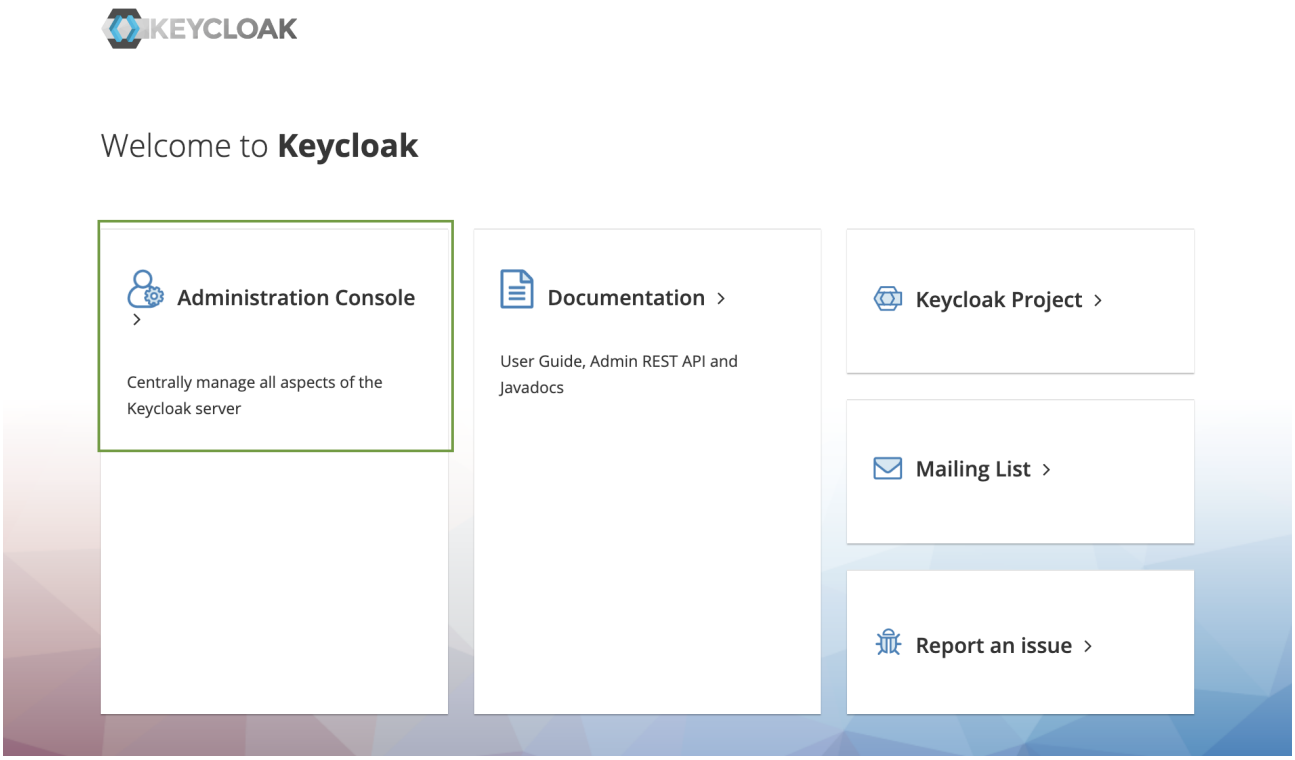
For more detailed information, refer to the official Keycloak documentation:

» Keycloak documentation

Creating a new realm

A realm is a space where you manage objects, including users, applications, roles, and groups. To create a new realm:

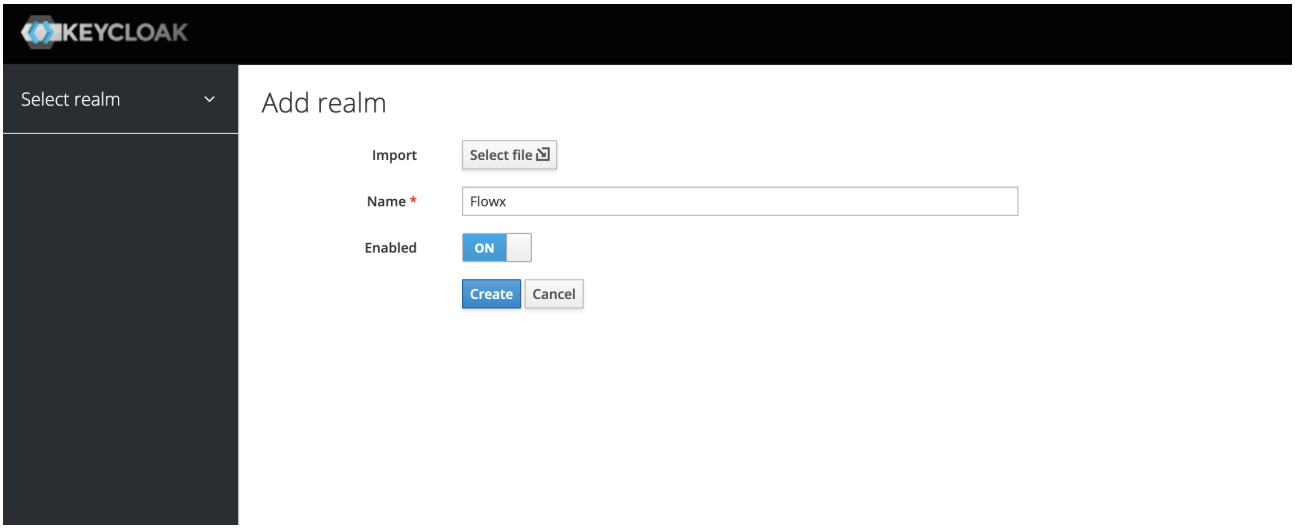
1. Log in to the **Keycloak Admin Console** using the appropriate URL for your environment (e.g., QA, development, production).



2. In the top left corner dropdown menu, click **Add Realm**.

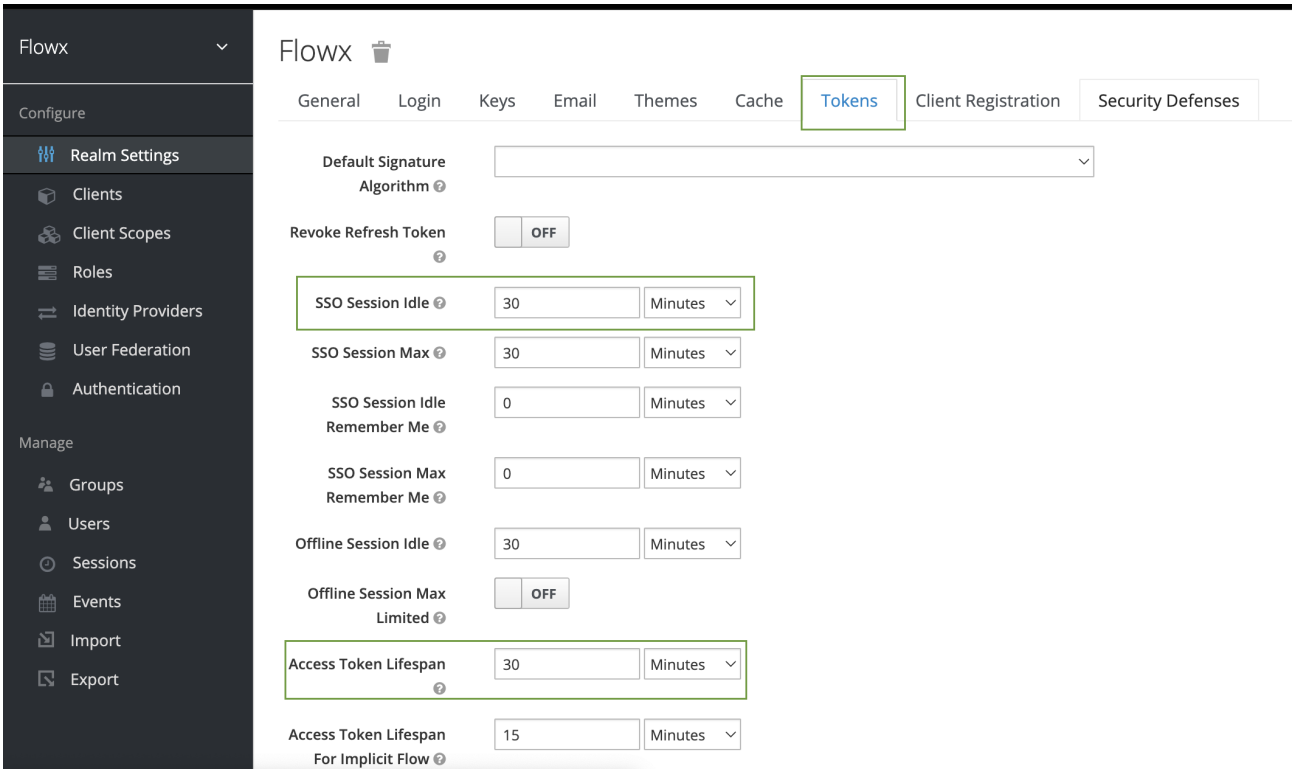
> **INFO**
>
> If you are logged in to the master realm this dropdown menu lists all the realms created. The **Add Realm** page opens.

3. Enter a realm name and click Create.



4. Configure the realm settings (**Realm Settings** → **Tokens**), such as SSO session idle and access token lifespan, according to your organization's needs:

- **SSO Session idle** - suggested: **30 Minutes**
- **Access Token Lifespan** - suggested: **30 Minutes**



Creating/importing user groups and roles

You can either create or import a user group into a realm. We prepared a script that helps you to import a **super admin group** provided with the necessary **default user roles**.

You can create or import user groups into a realm. If you choose to import, follow the provided **script** to import a **super admin group**(`SUPER_ADMIN_USERS`) with **default user roles**. After importing, add an admin user to the group and assign the necessary roles.

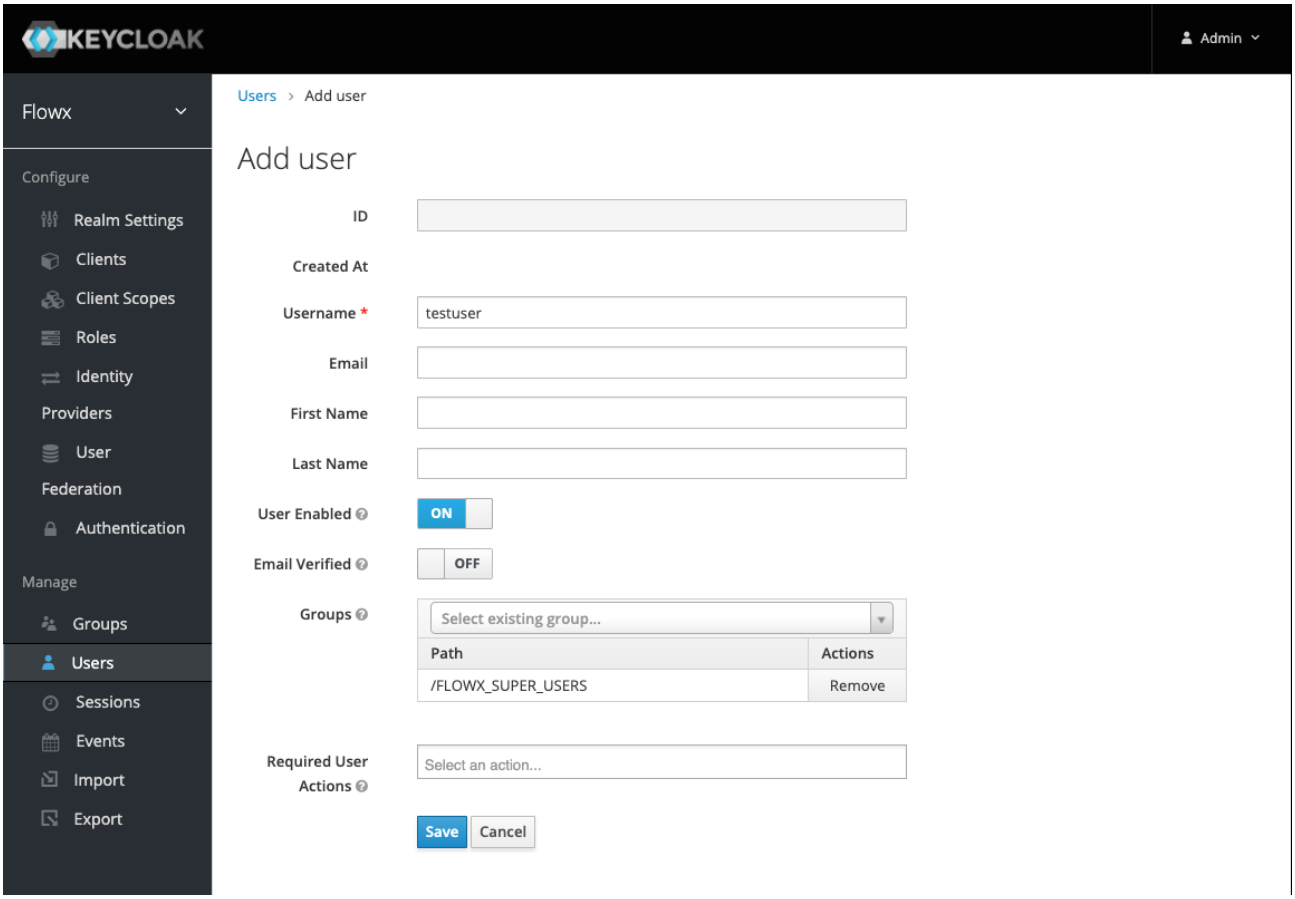
Make sure to validate the imported roles by checking the following section:

» Default roles

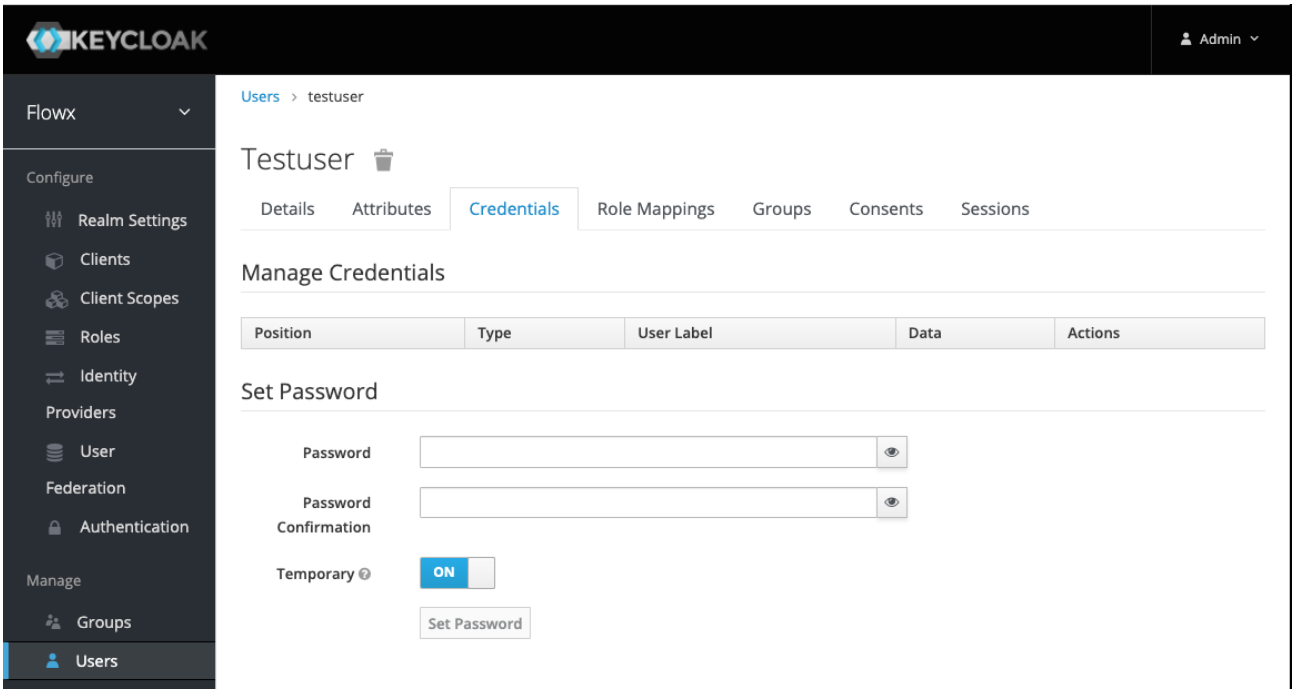
Creating new users

To create a new user in a realm and generate a temporary password:

1. In the left menu bar, click **Users** to open the user list page.
2. On the right side of the empty user list, click **Add User**.
3. Fill in the required fields, including the **username**, and ensure **Email Verified** is set to **ON**.
4. In the **Groups** field, choose a group from the dropdown menu, in our case: `FLOWX_SUPER_USERS`.



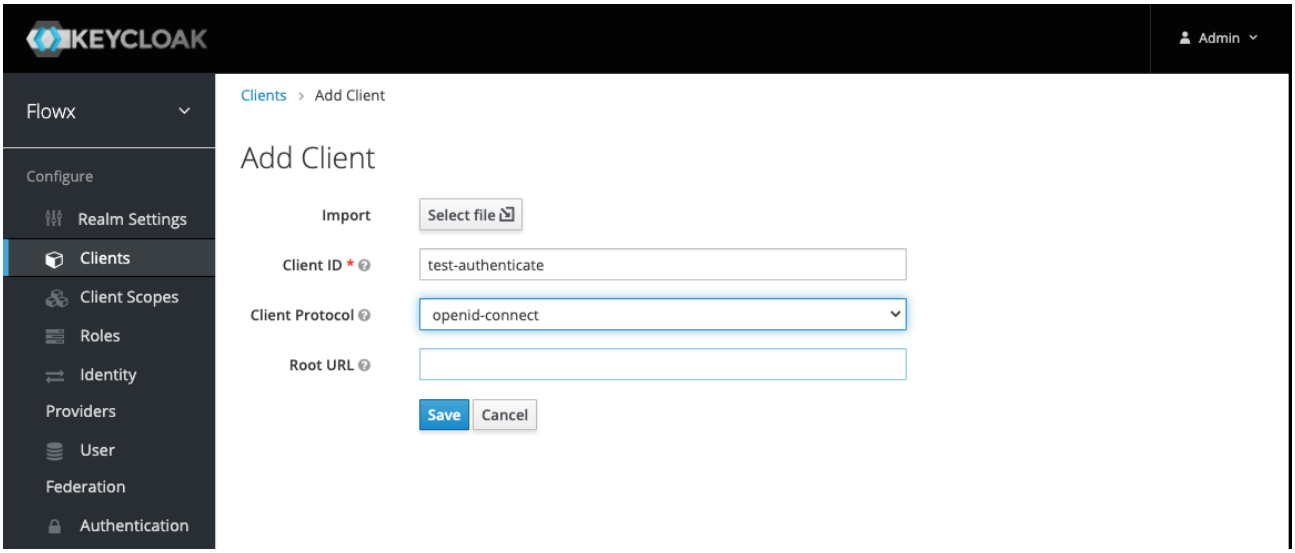
5. Save the user, go to the **Credentials** tab, and set a temporary password.



Adding clients

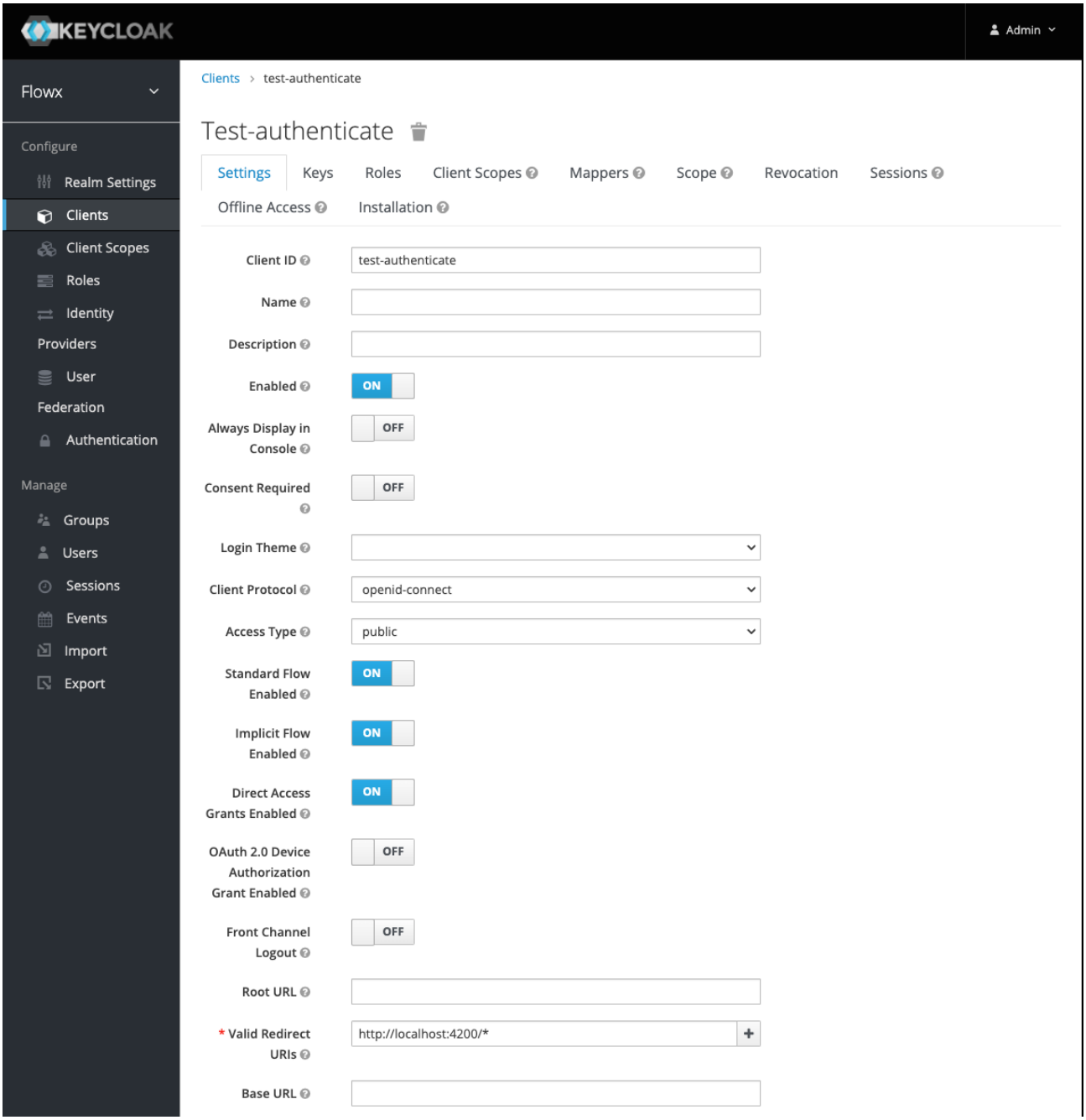
Clients represent trusted browser apps and web services in a realm. To add clients:

1. Click **Clients** in the top left menu, then click **Create**.
2. Set a client ID as `{example}-authenticate`, which will be used for login, logout, and refresh token operations.
3. Set the **Client Protocol** type as `openid-connect`.



3. Open the newly created **client** and edit the following properties:

- Set **Access type** to **public** (this will not require a secret)
- Set **Valid redirect URIs**, specifying a valid URI pattern that a browser can redirect to after a successful login or logout, simple wildcards are allowed
- Enable **Direct Access Grants** and **Implicit Flow** by setting them to **ON**.
- Switch **Backchannel Logout Session Required** to **OFF**



4. Add **mappers** to `{example}-authenticate` client.

> **INFO**

> Refer to the next section on how to add mappers and which mappers to clients.

Adding protocol mappers

Protocol mappers in Keycloak allow for the transformation of tokens and documents, enabling actions such as mapping user data into protocol claims or modifying requests between clients and the authentication server.

To enhance your clients, consider adding the following mappers:

- Group Membership mapper - `realm-groups`: This mapper can be utilized to map user groups to the authorization token.
- User Attribute mapper - `business filter mapper`: Use this mapper to map custom attributes, for example, mapping the businessFilters list, to the token claim.
- User Realm role - `realm-roles`: This mapper enables mapping a user's realm role to a token claim.

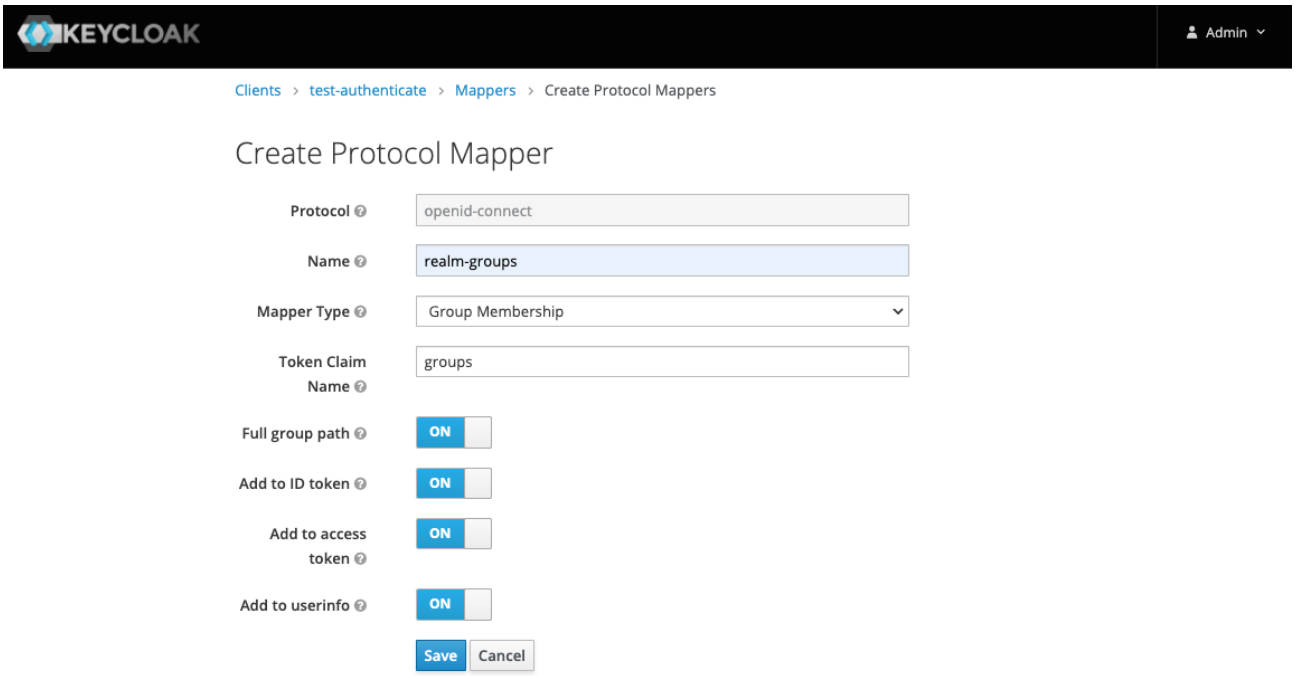
By incorporating these mappers, you can further customize and enrich the information contained within your tokens.

Group Membership mapper

To add a group membership mapper:

1. Navigate to **Clients** and select your desired client, in our case, `{example}-authenticate`

2. Go to the **Mappers** tab and click **Create** to create a new mapper.
3. Provide a descriptive **Name** for the mapper to easily identify its purpose.
4. Select **Group Membership** as the mapper type.
5. Set the token claim name for including groups in the token. In this case, set it as `groups`.



By configuring the group membership mapper, you will be able to include the user's group information in the token for authorization purposes.

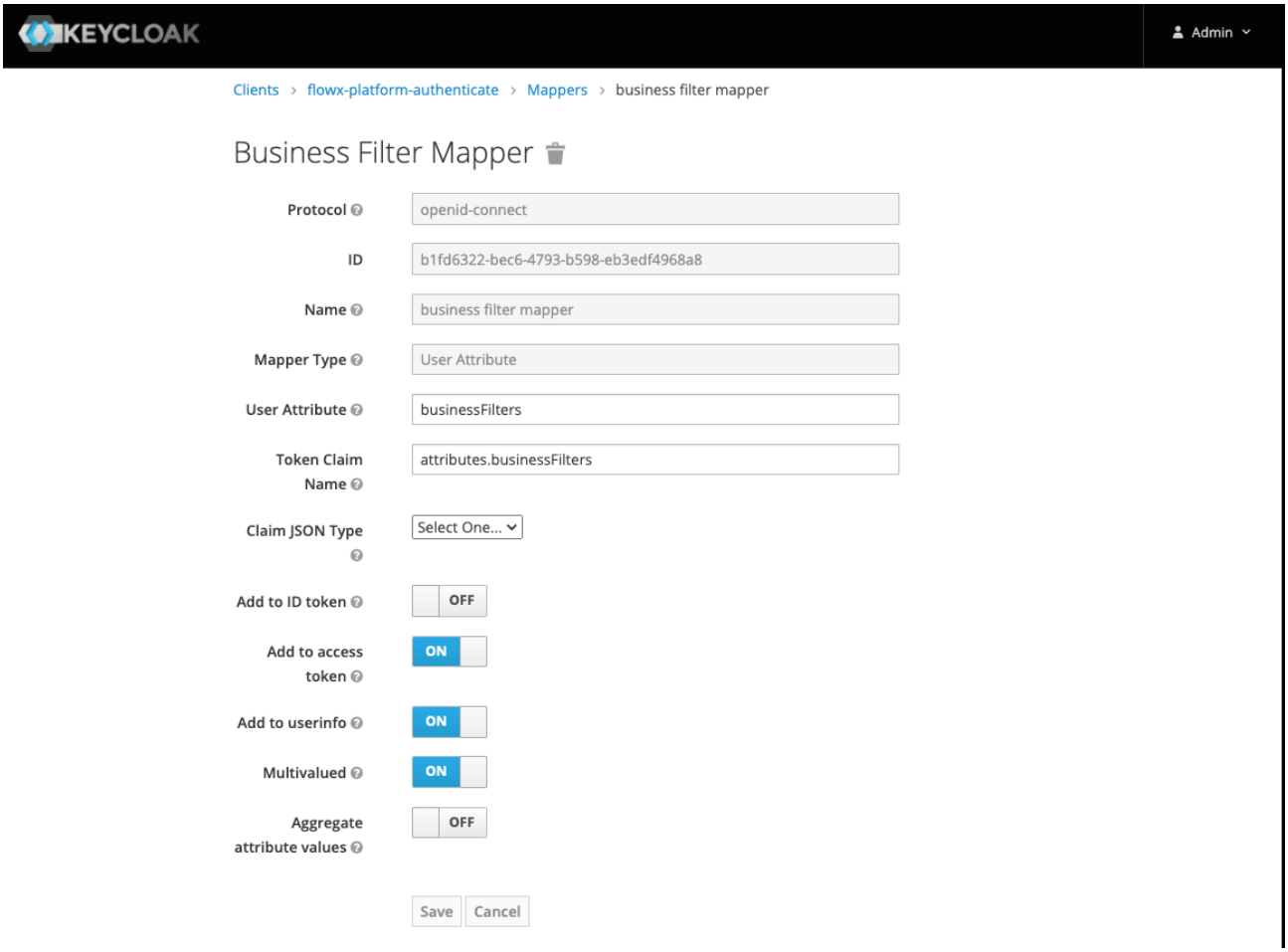
User Attribute mapper

To include custom attributes such as **business filters** in the token claim, you can add a user attribute mapper with the following settings:

1. Go to the desired client, `{example}-authenticate`, and navigate to the Mappers section.

2. Click on **Create** to create a new mapper.
3. Configure the following settings for the user attribute mapper:

- **Mapper Type**: User Attribute
- **User Attribute**: businessFilters
- **Token Claim Name**: attirubtes.businessFilters
- **Add to ID token**: OFF
- **Multivalued**: ON



By adding this user attribute mapper, the custom attribute "businessFilters" will be included in the token claim under the name "attributes.businessFilters". This will

allow you to access and utilize the business filters information within your application.

You can find more information about business filters in the following section:

» Business filters

User realm role

Add **roles mapper** to `{example}-authenticate` client - so roles will be available on the OAuth user info response.

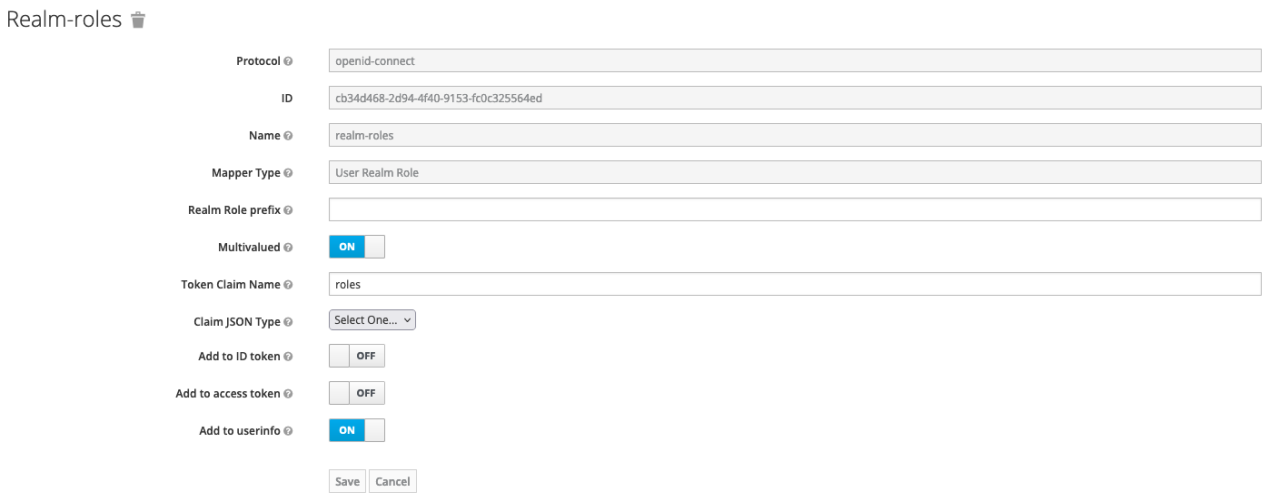
To add a roles mapper, follow these steps:

1. Go to the desired client, `{example}-authenticate`, and navigate to the Mappers section.
2. Click on **Create** to create a new mapper.
3. Configure the following settings for the user attribute mapper:

- **Mapper Type**: User Realm Role
- **Token Claim Name**: role
- **Add to userinfo**: ON

By adding this roles mapper, the assigned realm roles of the user will be available in the OAuth user info response under the claim name "roles". This allows you to access and utilize the user's realm roles within your application.

Please note that you can repeat these steps to add multiple roles mappers if you need to include multiple realm roles in the token claim.



Examples

Login

```
curl --location --request POST
'http://localhost:8080/realms/flowx/protocol/openid-
connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'username=admin@flowx.ai' \
--data-urlencode 'password=password' \
--data-urlencode 'client_id= example-authenticate'
```

Refresh token

```
curl --location --request POST
'http://localhost:8080/realms/flowx/protocol/openid-
connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=refresh_token' \
```

```
--data-urlencode 'client_id= example-authenticate' \
--data-urlencode 'refresh_token=ACCESS_TOKEN'
```

User info

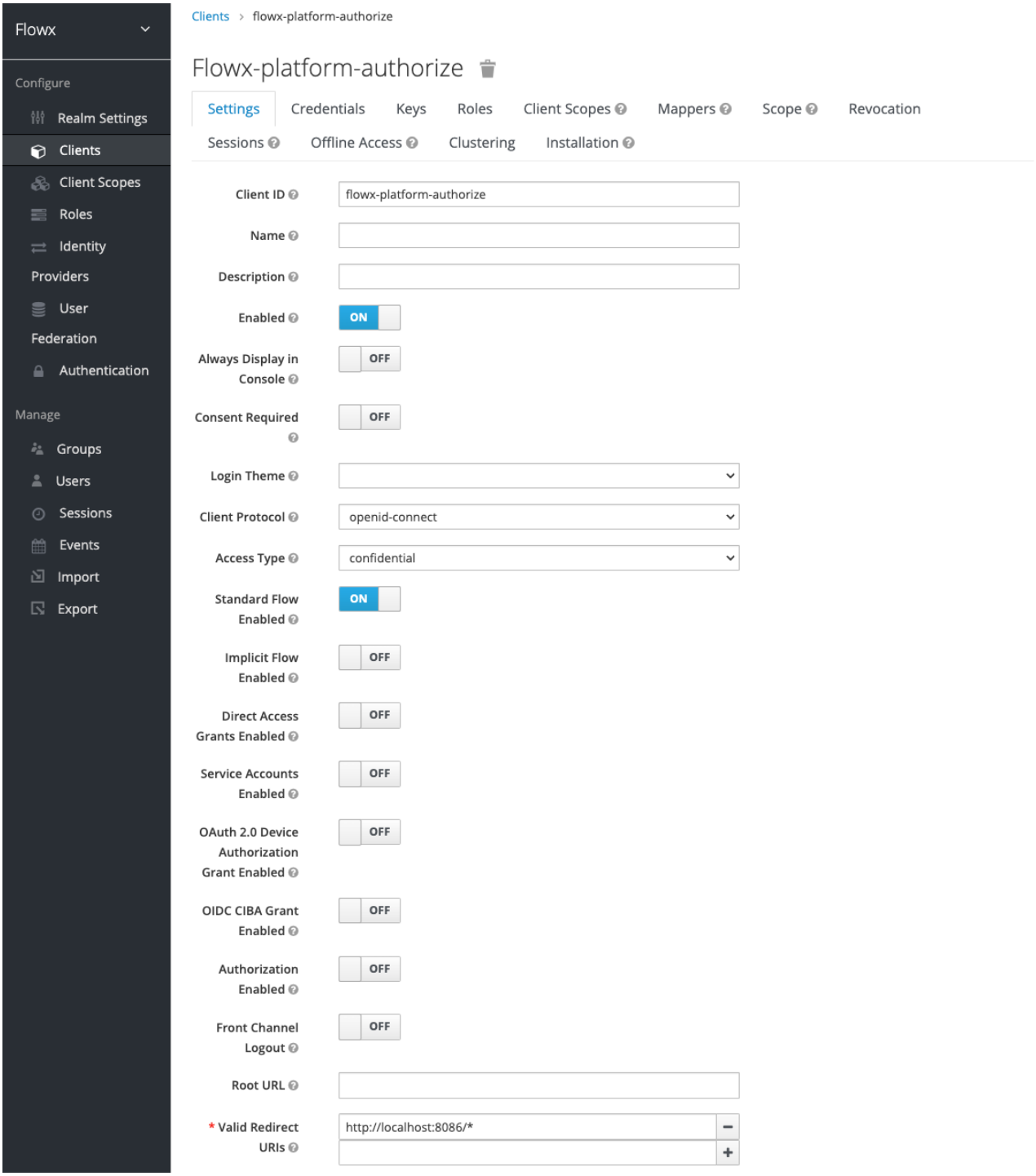
```
curl --location --request GET
'localhost:8080/realms/flowx/protocol/openid-
connect/userinfo' \
--header 'Authorization: Bearer ACCESS_TOKEN' \
```

Authorizing client

Add `{example}-platform-authorize` client - it will be used to authorize rest requests to microservices and Kafka

- set **Client Protocol** to **openid-connect**
- set **Access type** as **confidential**
- disable **Direct Access Grants Enabled** - OFF
- **Valid Redirect URIs** - mandatory
- disable **Backchannel Logout Session Required** - OFF

Once you have configured these settings, the `{example}-platform-authorize` client will be created and can be used to authorize REST requests to microservices and Kafka within your application.



Minimal auth config for microservices

```
security:
  type: oauth2
  basic:
    enabled: false
  oauth2:
    base-server-url: http://localhost:8080
    realm: flowx
    client:
      access-token-uri: ${security.oauth2.base-server-
url}/realms/${security.oauth2.realm}/protocol/openid-connect/t
      client-id: example-authorize
      client-secret: CLIENT_SECRET
    resource:
      user-info-uri: ${security.oauth2.base-server-
url}/realms/${security.oauth2.realm}/protocol/openid-connect/u
```

Adding service accounts

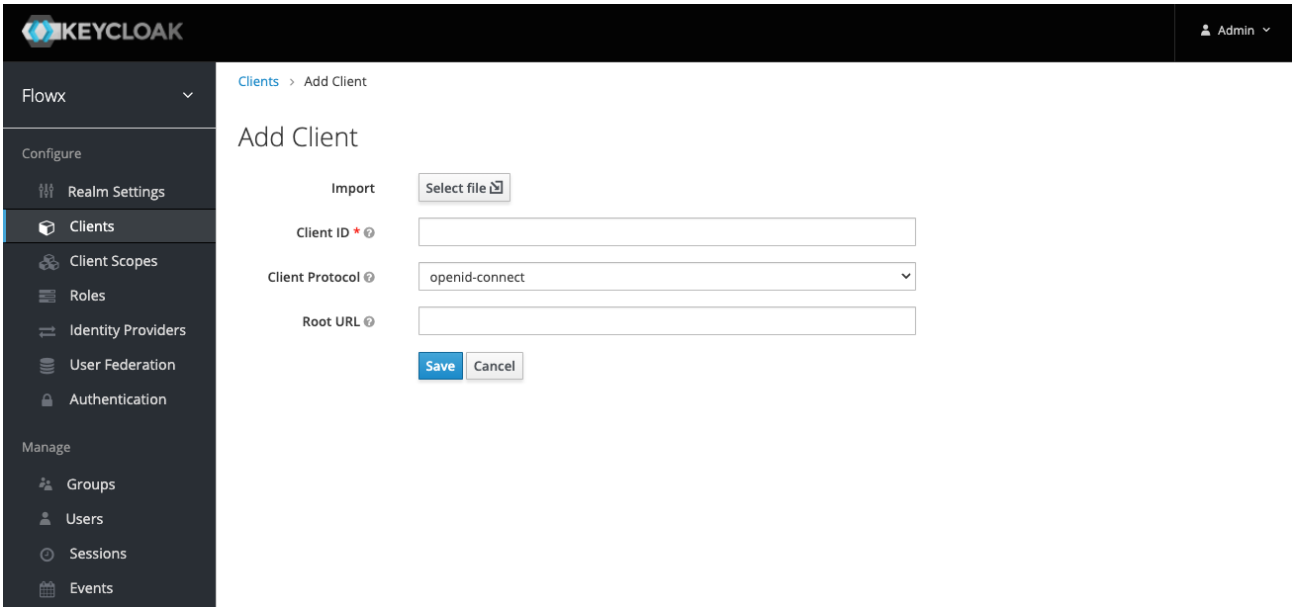
> **INFO**
>
> **What is a service account?**
>
> A service account is an account that grants direct access to the Keycloak API for a specific component.

Admin service account

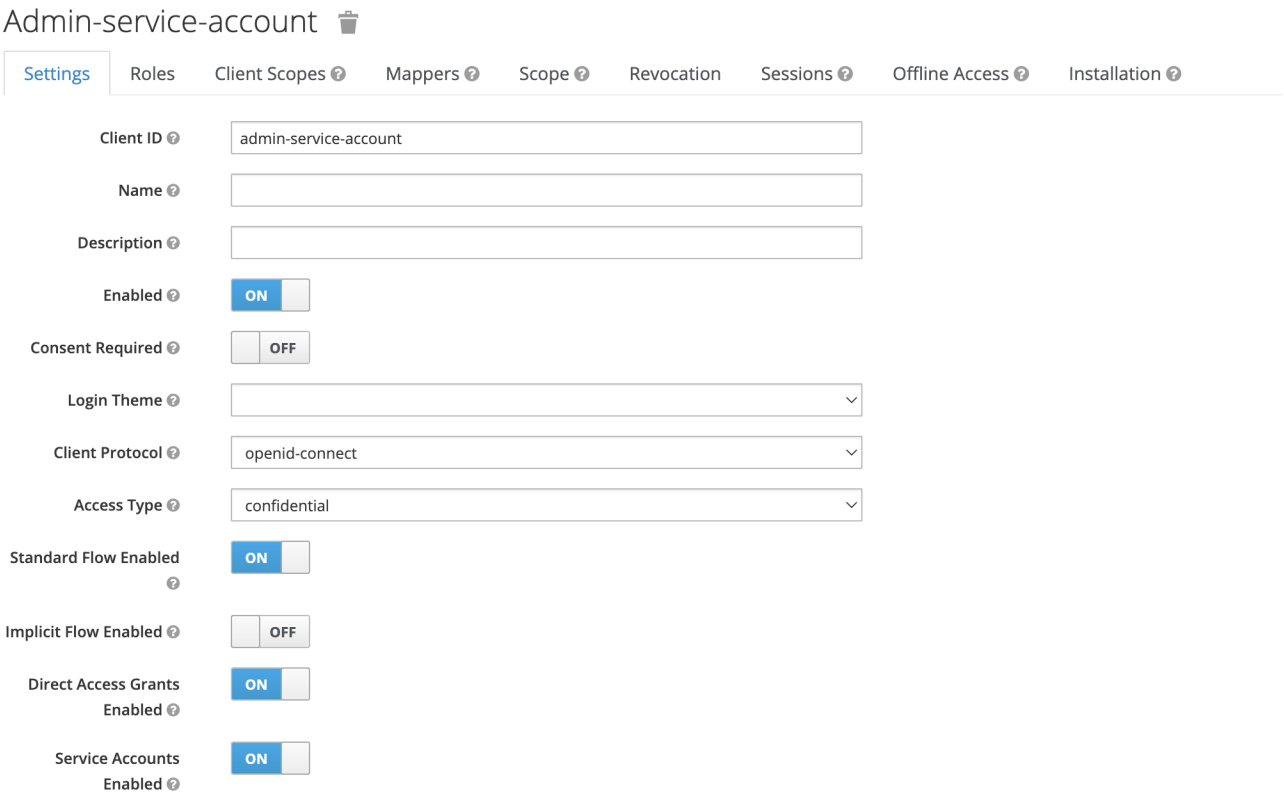
The admin microservice requires an admin service account to make direct calls to the Keycloak API.

Follow these steps to add an **admin service account**:

1. Add a new client by selecting **Clients** then click **Create**.



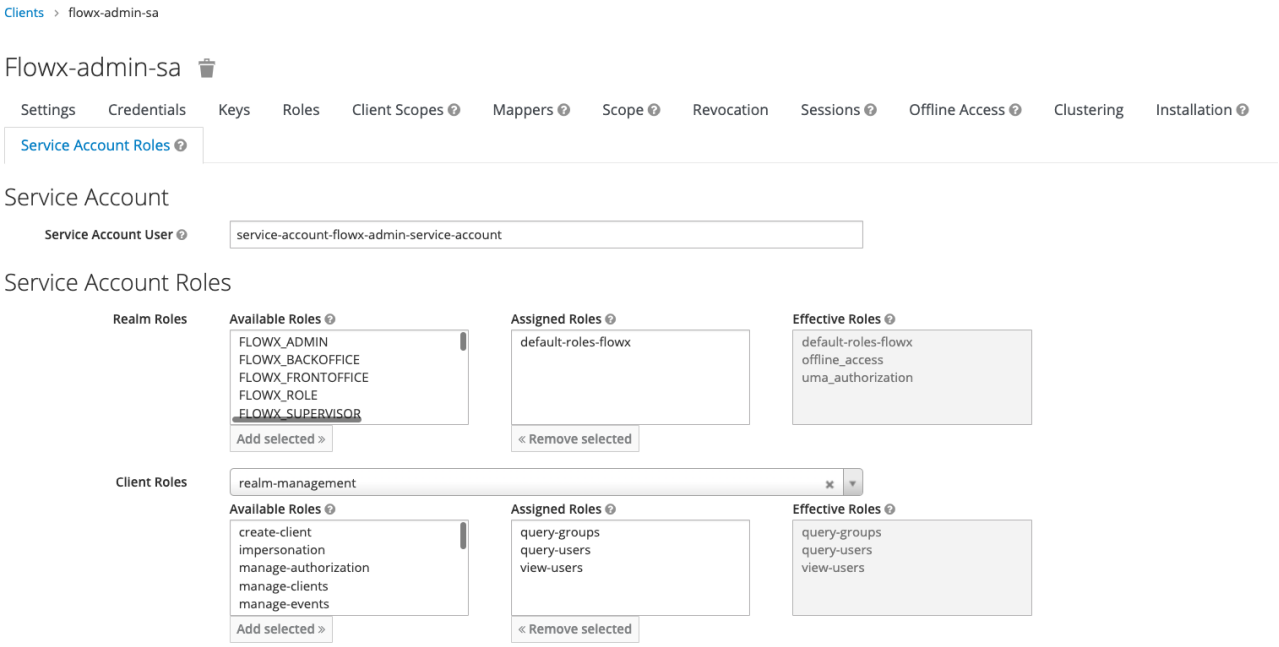
2. Next, set **Access type** as **confidential** and enable **Service Accounts**.



3. Go to **Clients** → **realm-management** → **Roles** and add the following **service account client roles** under **realm-management**:

- **view-users**
- **query-groups**
- **query-users**

4. Assign the necessary **service account roles**:



In the provided example, the **admin service account** can have the following assigned roles, depending on the required access scopes:

- **manage-users**
- **query-users**
- **manage-realm**

> **INFO**
>
> The admin service account does not require mappers as it doesn't utilize roles. Service account roles include client roles from the `realm-management`.

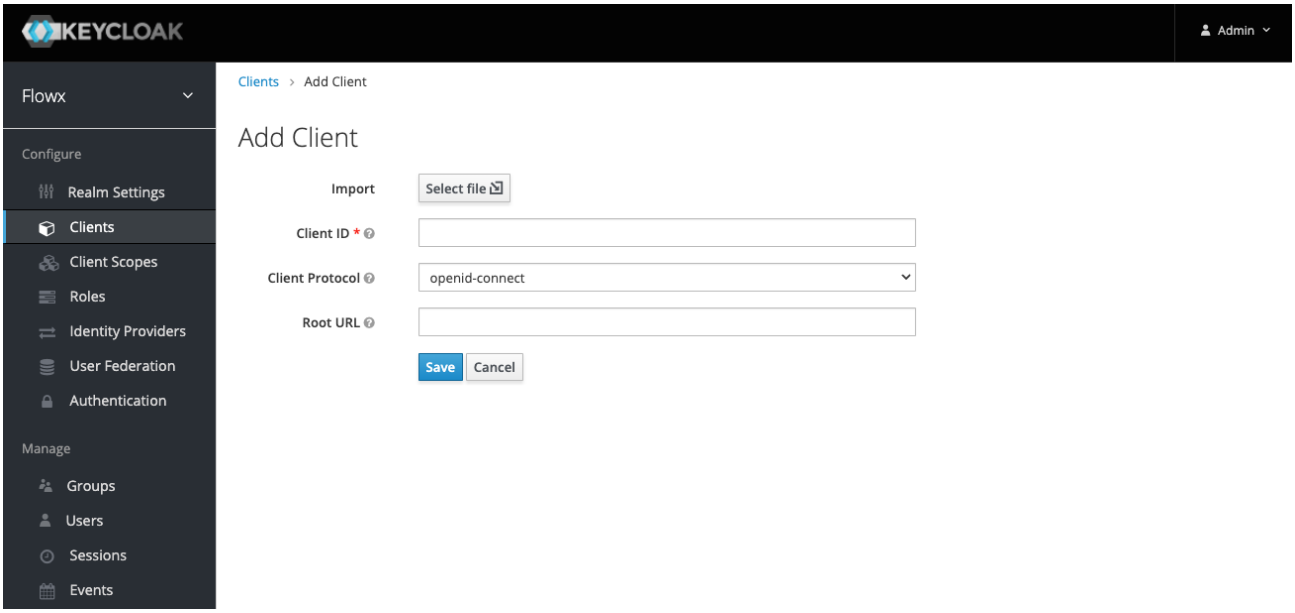
For detailed information, refer to the following section:

» Configuring access rights for admin

Task management service account

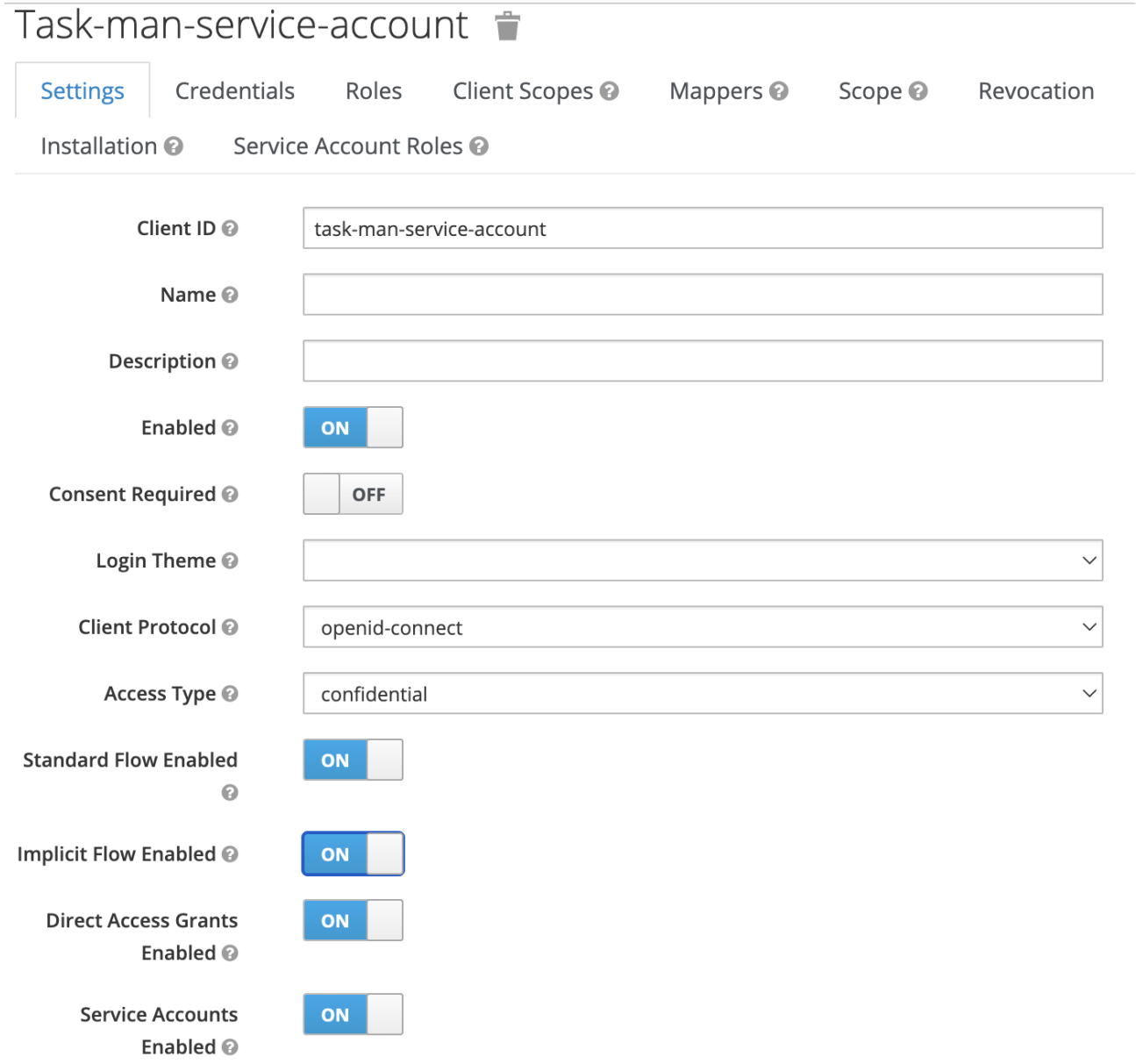
The task management microservice requires a service account to make direct calls to the Keycloak API. Follow these steps to add a task management service account:

1. Add a new client by selecting **Clients** then click **Create**.



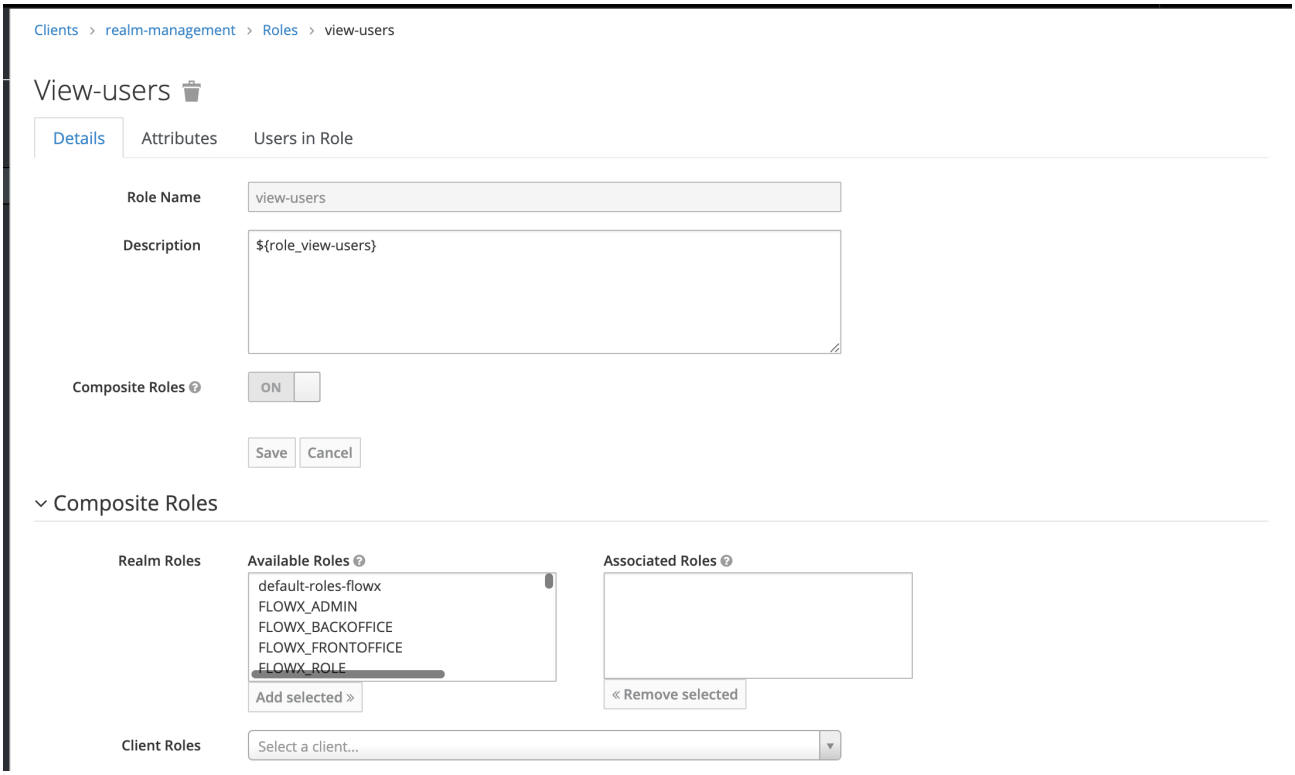
2. Next, set the following properties:

- **Access type** - confidential
- **Service Accounts Enabled** - ON

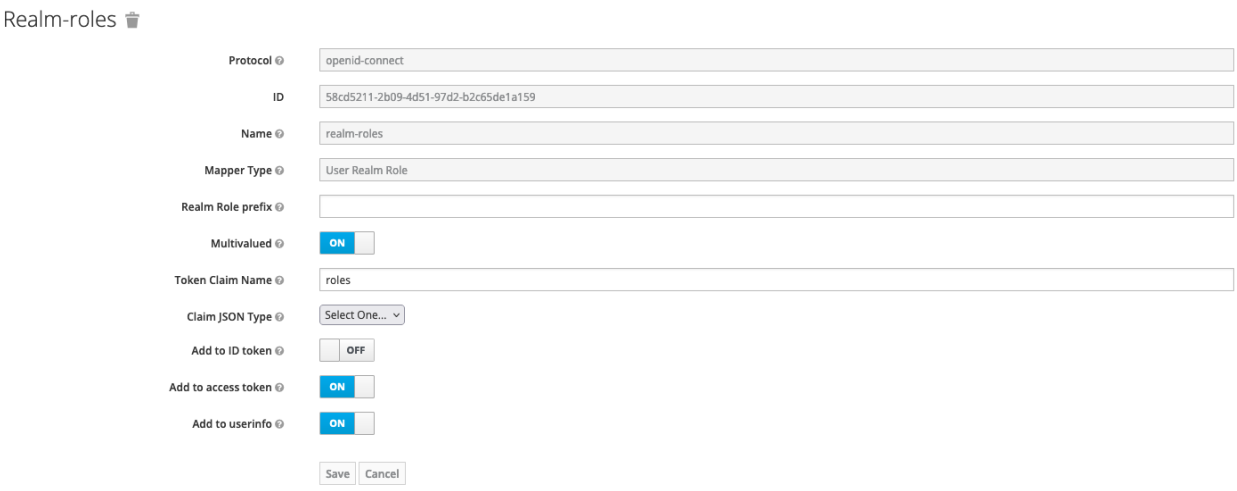


3. Go to **Clients** → **realm-management** → **Roles** and add the following **service account client roles**:

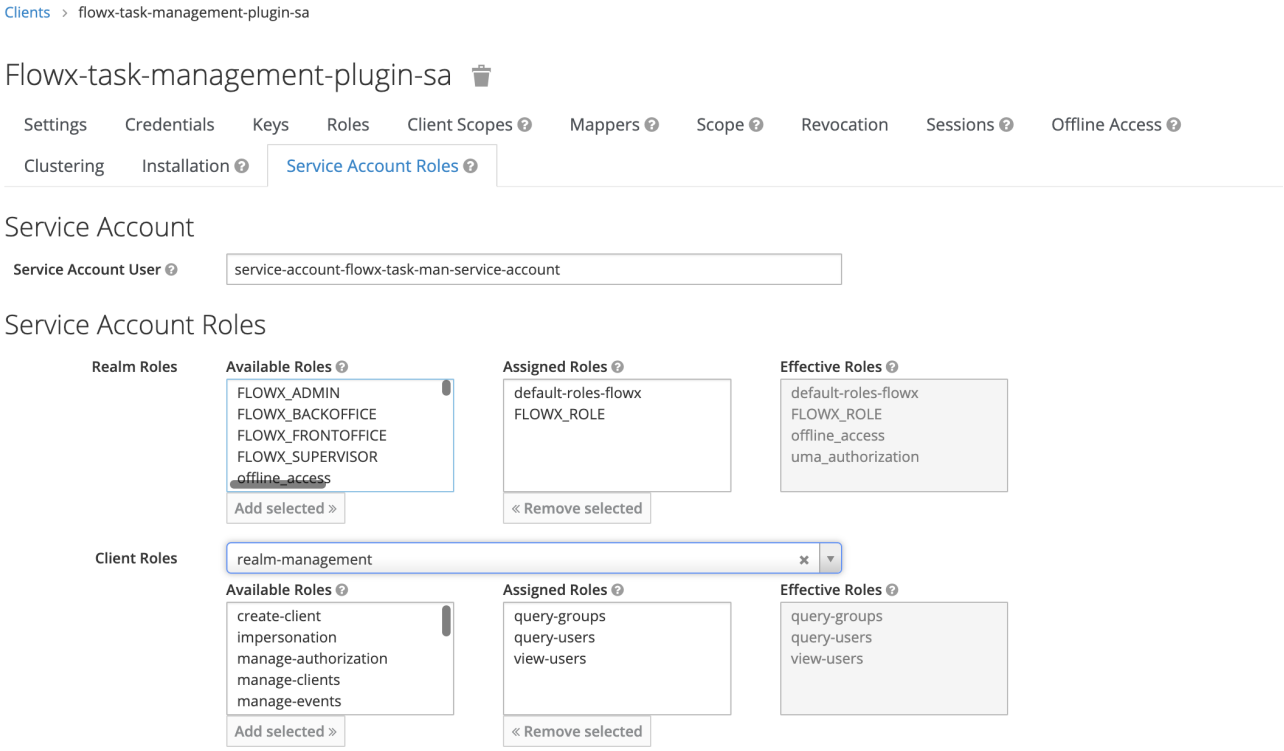
- **view-users**
- **query-groups**
- **query-users**



4. Configure a **realm roles mapper**:



5. Assign the necessary service account roles, including `FLOWX_ROLE`.



In the provided example, the **task management service account** can have the following assigned roles, depending on the required access scopes:

- **view-users**
- **query-groups**
- **query-users**

For more information, check the following section:

» Configuring access rights for Task Management

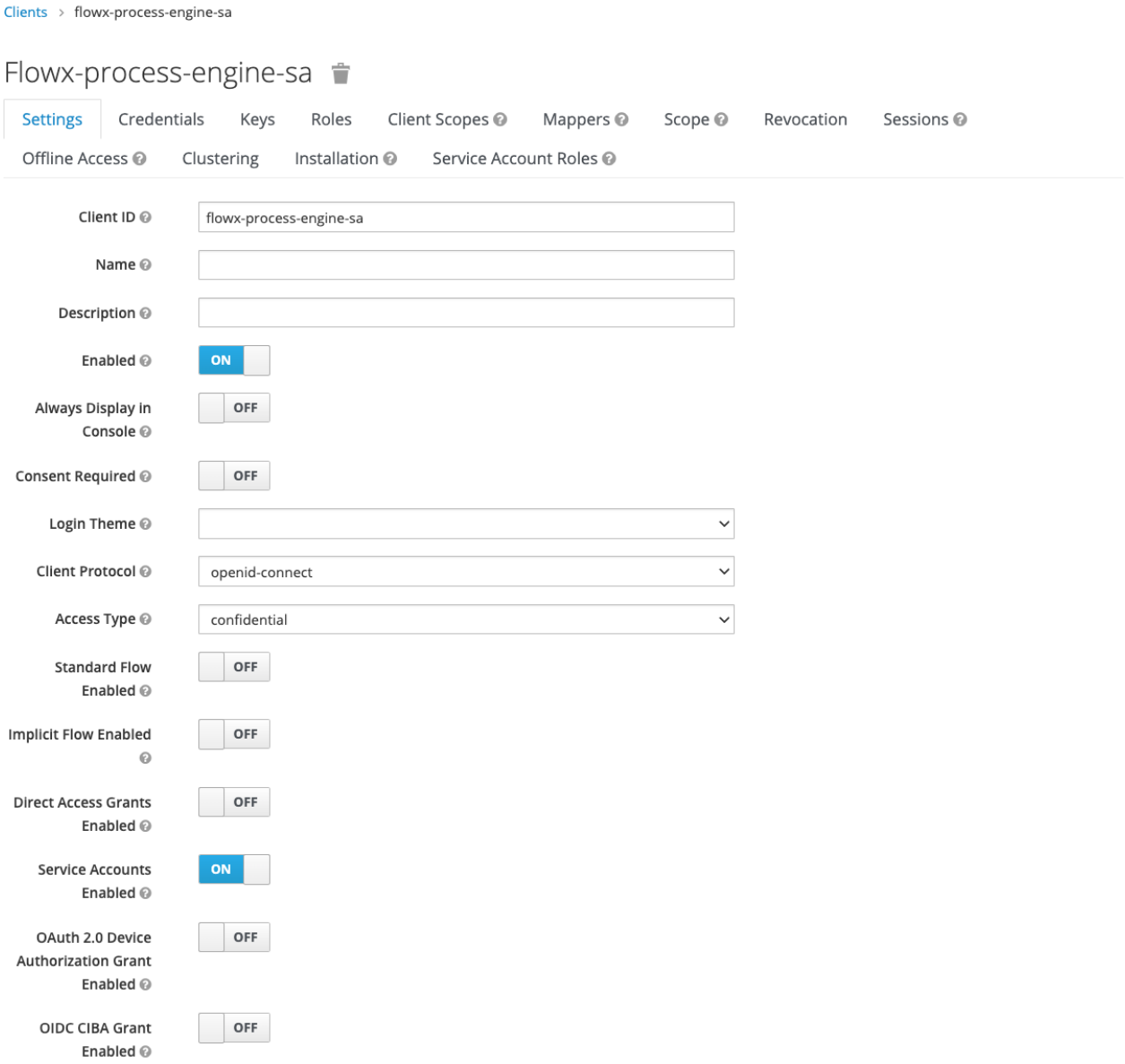
Process engine service account

The process engine requires a process engine service account to make direct calls to the Keycloak API.

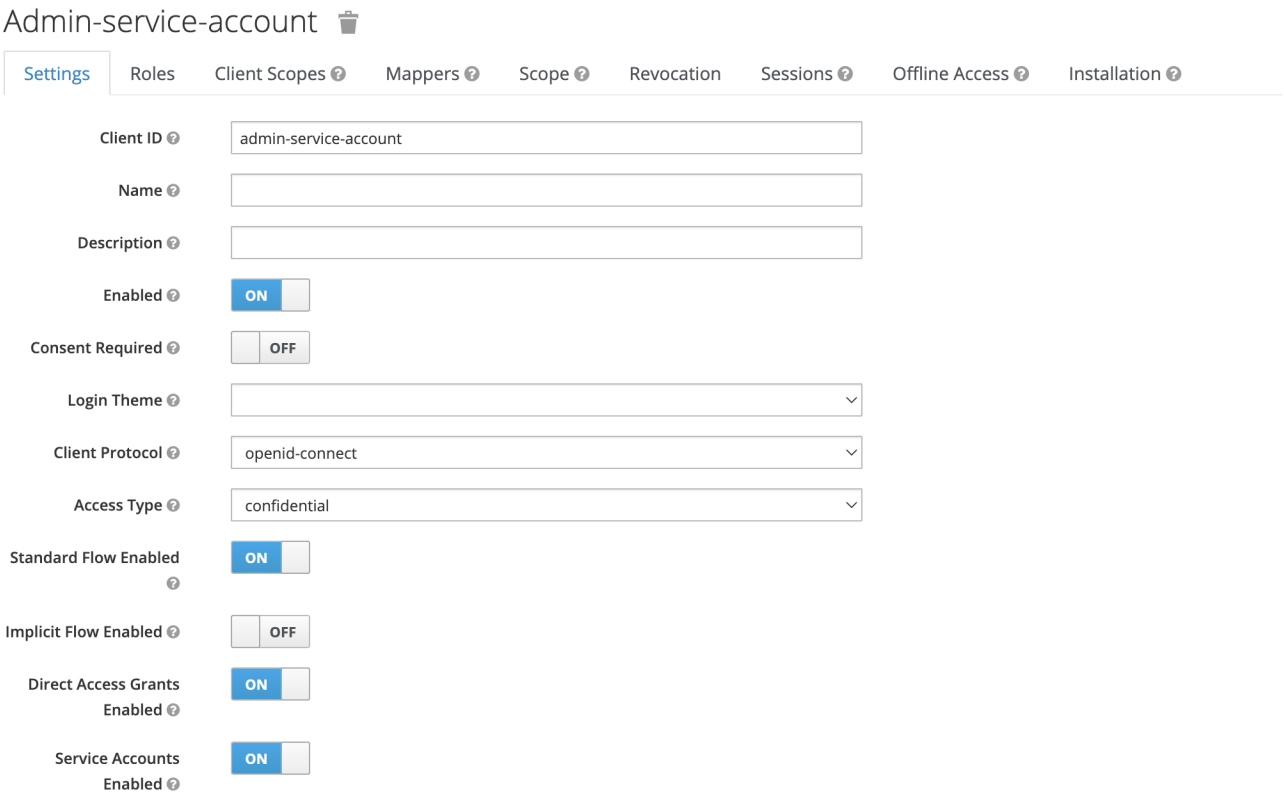
> **INFO**
>
> This service account is needed so the use of Start Catch Event node is possible.

Follow these steps to add a **process engine service account**:

1. Add a new client by selecting **Clients** then click **Create**.

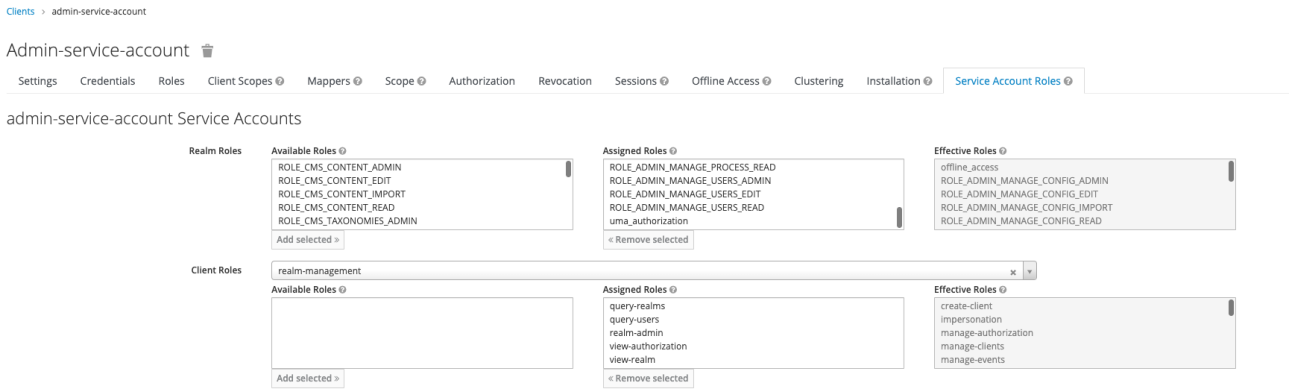


2. Next, set **Access type** as **confidential** and enable **Service Accounts**.



> **INFO**
>
> This service account does not require client roles.

3. Assign the necessary service account roles, including `FLOWX_ROLE`.



Was this page helpful?

PLATFORM SETUP GUIDES / Access management / Default roles

Below you can find the list of all the default roles that you can add or import into the Identity and Access Management solution to properly manage the access to all the FLOWX.AI microservices.

Default roles

A complete list of all the default roles based on modules (access scope):

Module	Scopes	Role default value

Module	Scopes	Role default value
manage-platform	read	ROLE_ADMIN_MANAGE_PLATFORM_READ
manage-platform	admin	ROLE_ADMIN_MANAGE_PLATFORM_ADMIN
manage-processes	import	ROLE_ADMIN_MANAGE_PROCESS_IMPORT
manage-processes	read	ROLE_ADMIN_MANAGE_PROCESS_READ
manage-processes	edit	ROLE_ADMIN_MANAGE_PROCESS_EDIT
manage-processes	admin	ROLE_ADMIN_MANAGE_PROCESS_ADMIN
manage-integrations	admin	ROLE_ADMIN_MANAGE_INTEGRATIONS_ADMIN
manage-integrations	read	ROLE_ADMIN_MANAGE_INTEGRATIONS_READ
manage-integrations	edit	ROLE_ADMIN_MANAGE_INTEGRATIONS_EDIT

Module	Scopes	Role default value
manage-integrations	import	ROLE_ADMIN_MANAGE_INTEGRATIONS_IMPOR
manage-configurations	import	ROLE_ADMIN_MANAGE_CONFIG_IMPORT
manage-configurations	read	ROLE_ADMIN_MANAGE_CONFIG_READ
manage-configurations	edit	ROLE_ADMIN_MANAGE_CONFIG_EDIT
manage-configurations	admin	ROLE_ADMIN_MANAGE_CONFIG_ADMIN
manage-users	read	ROLE_ADMIN_MANAGE_USERS_READ
manage-users	edit	ROLE_ADMIN_MANAGE_USERS_EDIT
manage-users	admin	ROLE_ADMIN_MANAGE_USERS_ADMIN
manage-processes	edit	ROLE_ENGINE_MANAGE_PROCESS_EDIT

Module	Scopes	Role default value
manage-processes	admin	ROLE_ENGINE_MANAGE_PROCESS_ADMIN
manage-instances	read	ROLE_ENGINE_MANAGE_INSTANCE_READ
manage-instances	admin	ROLE_ENGINE_MANAGE_INSTANCE_ADMIN
manage-licenses	read	ROLE_LICENSE_MANAGE_READ
manage-licenses	edit	ROLE_LICENSE_MANAGE_EDIT
manage-licenses	admin	ROLE_LICENSE_MANAGE_ADMIN
manage-contents	import	ROLE_CMS_CONTENT_IMPORT
manage-contents	read	ROLE_CMS_CONTENT_READ
manage-contents	edit	ROLE_CMS_CONTENT_EDIT

Module	Scopes	Role default value
manage-contents	admin	ROLE_CMS_CONTENT_ADMIN
manage-media-library	import	ROLE_MEDIA_LIBRARY_IMPORT
manage-media-library	read	ROLE_MEDIA_LIBRARY_READ
manage-media-library	edit	ROLE_MEDIA_LIBRARY_EDIT
manage-media-library	admin	ROLE_MEDIA_LIBRARY_ADMIN
manage-taxonomies	import	ROLE_CMS_TAXONOMIES_IMPORT
manage-taxonomies	read	ROLE_CMS_TAXONOMIES_READ
manage-taxonomies	edit	ROLE_CMS_TAXONOMIES_EDIT
manage-taxonomies	admin	ROLE_CMS_TAXONOMIES_ADMIN

Module	Scopes	Role default value
manage-tasks	read	ROLE_TASK_MANAGER_TASKS_READ
manage-hooks	import	ROLE_TASK_MANAGER_HOOKS_IMPORT
manage-hooks	read	ROLE_TASK_MANAGER_HOOKS_READ
manage-hooks	edit	ROLE_TASK_MANAGER_HOOKS_EDIT
manage-hooks	admin	ROLE_TASK_MANAGER_HOOKS_ADMIN
manage-process-allocation-settings	import	ROLE_TASK_MANAGER_PROCESS_ALLOCATIO
manage-process-allocation-settings	read	ROLE_TASK_MANAGER_PROCESS_ALLOCATIO

Module	Scopes	Role default value
manage-process-allocation-settings	edit	ROLE_TASK_MANAGER_PROCESS_ALLOCATIO
manage-process-allocation-settings	admin	ROLE_TASK_MANAGER_PROCESS_ALLOCATIO
manage-out-of-office-users	import	ROLE_TASK_MANAGER_OOO_IMPORT
manage-out-of-office-users	read	ROLE_TASK_MANAGER_OOO_READ
manage-out-of-office-users	edit	ROLE_TASK_MANAGER_OOO_EDIT
manage-out-of-office-users	admin	ROLE_TASK_MANAGER_OOO_ADMIN

Module	Scopes	Role default value
manage-notification-templates	import	ROLE_NOTIFICATION_TEMPLATES_IMPORT
manage-notification-templates	read	ROLE_NOTIFICATION_TEMPLATES_READ
manage-notification-templates	edit	ROLE_NOTIFICATION_TEMPLATES_EDIT
manage-notification-templates	admin	ROLE_NOTIFICATION_TEMPLATES_ADMIN
manage-notifications	import	ROLE_MANAGE_NOTIFICATIONS_IMPORT
manage-notifications	read	ROLE_MANAGE_NOTIFICATIONS_READ
manage-notifications	edit	ROLE_MANAGE_NOTIFICATIONS_EDIT

Module	Scopes	Role default value
manage-notifications	admin	ROLE_MANAGE_NOTIFICATIONS_ADMIN
manage-document-templates	import	ROLE_DOCUMENT_TEMPLATES_IMPORT
manage-document-templates	read	ROLE_DOCUMENT_TEMPLATES_READ
manage-document-templates	edit	ROLE_DOCUMENT_TEMPLATES_EDIT
manage-document-templates	admin	ROLE_DOCUMENT_TEMPLATES_ADMIN

Importing roles

INFO

You can import a super admin group and its default roles in Keycloak using the following script file.

DOWNLOAD THE SCRIPT + ROLES:

Import Script

You need to edit the following script parameters:

- `baseAuthUrl`
- `username`
- `password`
- `realm`
- `the name of the group for super admins`

The requests package is needed in order to run the script. It can be installed with the following command:

```
pip3 install requests
```

The script can be run with the following command:

```
python3 importUsers.py
```

Was this page helpful?

PLATFORM SETUP GUIDES / Audit setup guide

Introduction

This guide will walk you through the process of setting up the Audit service and configuring it to meet your needs.

Infrastructure prerequisites

The Audit service requires the following components to be set up before it can be started:

- **Docker engine** - version 17.06 or higher
- **Kafka** - version 2.8 or higher
- **Elasticsearch** - version 7.11.0 or higher

Dependencies

The Audit service is built as a Docker image and runs on top of Kafka and Elasticsearch. Therefore, these services must be set up and running before starting the Audit service.

- **Kafka configuration**
- **Authorization & access roles**
- **Elastic search**
- **Logging**

Configuration

Configuring Kafka

To configure the Kafka server for the Audit service, set the following environment variables:

- `SPRING_KAFKA_BOOTSTRAP_SERVERS` - address of the Kafka server, it should be in the format "host:port"
- `SPRING_KAFKA_CONSUMER_GROUP_ID` - the consumer group ID to be used for the audit logs
- `KAFKA_CONSUMER_THREADS` - the number of Kafka consumer threads to be used for processing audit logs
- `KAFKA_TOPIC_AUDIT_IN` - the topic key for receiving audit logs

Configuring Elasticsearch

To configure Elasticsearch, set the following environment variables:

- `SPRING_ELASTICSEARCH_REST_URIS` - the URL(s) of one or more Elasticsearch nodes to connect to
- `SPRING_ELASTICSEARCH_REST_DISABLESSL` - a boolean value that determines whether SSL should be disabled for Elasticsearch connections
- `SPRING_ELASTICSEARCH_REST_USERNAME` - the username to use for basic authentication when connecting to Elasticsearch
- `SPRING_ELASTICSEARCH_REST_PASSWORD` - the password to use for basic authentication when connecting to Elasticsearch

- `SPRING_ELASTICSEARCH_INDEX_SETTINGS_DATASTREAM` (used if ES is used across all dev environments) - the index settings for the datastreams that will be created in Elasticsearch

Configuring logging

To control the log levels, set the following environment variables:

- `LOGGING_LEVEL_ROOT` - the log level for the root spring boot microservice logs
- `LOGGING_LEVEL_APP` - the log level for app-level logs

> **CAUTION**
>
> Make sure to overwrite the placeholders (where needed) with the appropriate values before starting the service.

Was this page helpful?

PLATFORM SETUP GUIDES / CMS setup guide / Configuring access rights for CMS

Granular access rights can be configured for restricting access to the CMS component.

Two different access authorizations are provided, each with specified access scopes:

1. **Manage-contents** - for configuring access for manipulating CMS contents

Available scopes:

- import - users are able to import enumeration/substitution tags/ content models
- read - users are able to show enumeration/substitution tags/ content models, export enumeration/substitution tags/ content models
- edit - users are able to create/edit enumeration/substitution tags/ content models
- admin - users are able to delete enumeration/substitution tags/ content models

2. **Manage-taxonomies** - for configuring access for manipulating taxonomies

Available scopes

- read - users are able to show languages/source systems
- edit - users are able to edit languages/source systems
- admin - users are able to delete languages/source systems

3. **Manage-media-library** - for configuring access rights to use Media Library

Available scopes

- import - users are able to import assets
- read - users are able to view assets

- edit - users are able to edit assets
- admin - users are able to delete assets

The CMS service is preconfigured with the following default users roles for each of the access scopes mentioned above:

- **manage-contents**
  - import:
    - ROLE_CMS_CONTENT_IMPORT
    - ROLE_CMS_CONTENT_EDIT
    - ROLE_CMS_CONTENT_ADMIN
  - read:
    - ROLE_CMS_CONTENT_EDIT
    - ROLE_CMS_CONTENT_ADMIN
    - ROLE_CMS_CONTENT_READ
    - ROLE_CMS_CONTENT_IMPORT
  - edit:
    - ROLE_CMS_CONTENT_EDIT
    - ROLE_CMS_CONTENT_ADMIN
  - admin:
    - ROLE_CMS_CONTENT_ADMIN
- **manage-taxonomies**
  - import:
    - ROLE_CMS_TAXONOMIES_IMPORT
    - ROLE_CMS_TAXONOMIES_EDIT
    - ROLE_CMS_TAXONOMIES_ADMIN
  - read:

        - ROLE_CMS_TAXONOMIES_READ
        - ROLE_CMS_TAXONOMIES_IMPORT
        - ROLE_CMS_TAXONOMIES_EDIT
        - ROLE_CMS_TAXONOMIES_ADMIN
    - edit:
        - ROLE_CMS_TAXONOMIES_EDIT
        - ROLE_CMS_TAXONOMIES_ADMIN
    - admin:
        - ROLE_CMS_TAXONOMIES_ADMIN
- **manage-media-library**
    - import:
        - ROLE_MEDIA_LIBRARY_IMPORT
        - ROLE_MEDIA_LIBRARY_EDIT
        - ROLE_MEDIA_LIBRARY_EDIT
    - read:
        - ROLE_MEDIA_LIBRARY_READ
        - ROLE_MEDIA_LIBRARY_EDIT
        - ROLE_MEDIA_LIBRARY_ADMIN
        - ROLE_MEDIA_LIBRARY_IMPORT
    - edit:
        - ROLE_MEDIA_LIBRARY_EDIT
        - ROLE_MEDIA_LIBRARY_ADMIN
    - admin:
        - ROLE_MEDIA_LIBRARY_ADMIN

CAUTION

> The needed roles should be defined in the chosen identity provider solution.

In case other custom roles are needed, you can configure them using environment variables. More than one role can be set for each access scope.

To configure access for each of the roles above, adapt the following input:

`SECURITY_ACCESSAUTHORIZATIONS_AUTHORIZATIONNAME_SCOPES_SCOPENAME_ROLESALLOWED: NEEDED_ROLE_NAMES`

Possible values for AUTHORIZATIONNAME: MANAGECONTENTS, MANAGETAXONOMIES.

Possible values for SCOPENAME: import, read, edit, admin.

For example, if you need to configure role access for import, insert this:

```
SECURITY_ACCESSAUTHORIZATIONS_MANAGECONTENTS_SCOPES_IMPORT_ROLE
ROLE_CMS_CONTENT_IMPORT
```

Was this page helpful?

PLATFORM SETUP GUIDES / Events gateway setup guide

Introduction

This guide will walk you through the process of setting up the events-gateway service.

Infrastructure prerequisites

Before proceeding with the setup, ensure that the following components have been set up:

- **Redis** - version 6.0 or higher
- **Kafka** - version 2.8 or higher

Dependencies

- **Kafka** - used for event communication
- **Redis** - used for caching

Configuration

Configuring Kafka

Set the following Kafka-related configurations using environment variables:

- `SPRING_KAFKA_BOOTSTRAP_SERVERS` - the address of the Kafka server, it should be in the format "host:port"

Groupd IDs

The configuration parameters "KAFKA*CONSUMER_GROUP_ID**" are used to set the consumer group name for Kafka consumers that consume messages from topics. Consumer groups in Kafka allow for parallel message processing by distributing the workload among multiple consumer instances. By configuring the consumer group ID, you can specify the logical grouping of consumers that work together to process messages from the same topic, enabling scalable and fault-tolerant message consumption in your Kafka application.

Configuration Parameter	
`KAFKA_CONSUMER_GROUP_ID_PROCESS_ENGINE_COMMANDS_MESSAGE`	e c m
`KAFKA_CONSUMER_GROUP_ID_PROCESS_ENGINE_COMMANDS_DISCONNECT`	e c d

Configuration Parameter	
`KAFKA_CONSUMER_GROUP_ID_PROCESS_ENGINE_COMMANDS_CONNECT`	e c c
`KAFKA_CONSUMER_GROUP_ID_PROCESS_TASK_COMMANDS`	t c m

Threads

The configuration parameters "KAFKA*CONSUMER_THREADS*" are utilized to specify the number of threads assigned to Kafka consumers for processing messages from topics. These parameters allow you to fine-tune the concurrency and parallelism of your Kafka consumer application, enabling efficient and scalable message consumption from Kafka topics.

Configuration Parameter	De va

Configuration Parameter	De va
KAFKA_CONSUMER_THREADS_PROCESS_ENGINE_COMMANDS_MESSAGE	10
KAFKA_CONSUMER_THREADS_PROCESS_ENGINE_COMMANDS_DISCONNECT	5
KAFKA_CONSUMER_THREADS_PROCESS_ENGINE_COMMANDS_CONNECT	5
KAFKA_CONSUMER_THREADS_TASK_COMMANDS	10
KAFKA_AUTH_EXCEPTION_RETRY_INTERVAL	10

Kafka topics related to process instances

Configuration Parameter	
`KAFKA_TOPIC_EVENTS_GATEWAY_PROCESS_INSTANCE_IN_MESSAGE`	`ai`
`KAFKA_TOPIC_EVENTS_GATEWAY_PROCESS_INSTANCE_IN_DISCONNECT`	`ai`
`KAFKA_TOPIC_EVENTS_GATEWAY_PROCESS_INSTANCE_IN_CONNECT`	`ai`

Kafka topics related to tasks

Configuration Parameter	
`KAFKA_TOPIC_EVENTS_GATEWAY_TASK_IN_MESSAGE`	`ai.flowx.eventsga`

Configuring authorization & access roles

Set the following environment variables to connect to the identity management platform:

Configuration Parameter	Description
`SECURITY_OAUTH2_BASE_SERVER_URL`	Base URL of the OAuth2 server
`SECURITY_OAUTH2_CLIENT_CLIENT_ID`	Client ID for OAuth2 authentication

Configuration Parameter	Description
`SECURITY_OAUTH2_CLIENT_CLIENT_SECRET`	Client secret for OAuth2 authentication
`SECURITY_OAUTH2_REALM`	Realm for OAuth2 authentication

Redis

The process engine sends the messages to the events-gateway, which is responsible for sending them to Redis.

Configuration Parameter	Description
`SPRING_REDIS_HOST`	Hostname of the Redis server
`SPRING_REDIS_PASSWORD`	Password for Redis server
`SPRING_REDIS_TTL`	Time-to-live for Redis keys (default value:5000000 # milliseconds)

Master replica

The events-gateway can be configured to communicate with Redis using the MASTER_REPLICA replication mode by configuring the following property:

spring.redis.sentinel.nodes: replica1, replica2, replica3, etc...

Example

```
spring.redis.sentinel.nodes=host1:26379,host2:26379,host3:2637
```

In the above example, the Spring Boot application will connect to three Redis Sentinel nodes: host1:26379, host2:26379, and host3:26379.

The property value should be a comma-separated list of host:port pairs, where each pair represents the hostname or IP address and the port number of a Redis Sentinel node.

> **INFO**
>
> By default, Redis is standalone, so the configuration with `redis-replicas` is optional for high load use cases.

In the context of Spring Boot and Redis Sentinel integration, the `spring.redis.sentinel.nodes` property is used to specify the list of Redis Sentinel nodes that the Spring application should connect to. These nodes are responsible for monitoring and managing Redis instances.

Configuring logging

The following environment variables could be set in order to control log levels:

Configuration Parameter	Description

Configuration Parameter	Description
`LOGGING_LEVEL_ROOT`	Logging level for the root Spring Boot microservice logs
`LOGGING_LEVEL_APP`	Logging level for the application-level logs

Was this page helpful?

PLATFORM SETUP GUIDES / License engine setup guide / Configuring access rights for License

Granular access rights can be configured for restricting access to the License component.

The following access authorizations are provided, with the specified access scopes:

1. **Manage-licenses** - for configuring access for managing license related details

Available scopes:

- read - users are able to view the license report

- edit - users are able to update the license model and sync license data
- admin - users are able to download the license data

The License component is preconfigured with the following default users roles for each of the access scopes mentioned above:

- manage-licenses
  - read:
    - ROLE_LICENSE_MANAGE_READ
    - ROLE_LICENSE_MANAGE_EDIT
    - ROLE_LICENSE_MANAGE_ADMIN
  - edit:
    - ROLE_LICENSE_MANAGE_EDIT
    - ROLE_LICENSE_MANAGE_ADMIN
  - admin:
    - ROLE_LICENSE_MANAGE_ADMIN

> **DANGER**
>
> These roles need to be defined in the chosen identity provider solution.

In case other custom roles are needed, you can configure them using environment variables. More than one role can be set for each access scope.

To configure access for each of the roles above, adapt the following input:

```
SECURITY_ACCESSAUTHORIZATIONS_AUTHORIZATIONNAME_SCOPES_SCOPENAME_ROLESALLOWED: NEEDED_ROLE_NAMES
```

Possible values for `AUTHORIZATIONNAME: MANAGELICENSES`.

Possible values for `SCOPENAME`: read, edit, admin.

For example, if you need to configure role access for read, insert this:

```
SECURITY_ACCESSAUTHORIZATIONS_MANAGELICENSES_SCOPES_READ_ROLES/
ROLE_NAME_TEST
```

Was this page helpful?

PLATFORM SETUP GUIDES / License engine setup guide / Configuring access roles

> **CAUTION**
>
> Deprecated since platform version 1.16.0

The License engine is able to offer different levels of accessing license related information.

In order to restrict API calls by user role you will need to add the user roles in the application config. You can configure separate roles for the provided API base routes:

```
    - path: "/api/report"
      rolesAllowed: ${LICENSE_VIEW}
    - path: "/api/license-model"
      rolesAllowed: ${LICENSE_MANAGER}
    - path: "/api/sync/**"
      rolesAllowed: ${LICENSE_SUPER_MANAGER}
    - path: "/api/data/**"
      rolesAllowed: ${LICENSE_SUPER_USER}
```

- `LICENSE_VIEW` - users with this role will be able to view the status of the license (just the usage info, no extra details)
- `LICENSE_MANAGER` - users with this role will be able to configure the license
- `LICENSE_SUPER_MANAGER` - users with this role will be able to trigger sync for the existing license
- `LICENSE_SUPER_USER` - users with this role will be able to request a detailed report with details of custom identifiers and dates when they appear (this can contain personal data)

Was this page helpful?

PLATFORM SETUP GUIDES / Scheduler setup guide

Introduction

This guide will walk you through the process of setting up the Scheduler service using a Docker image.

Infrastructure prerequisites

- **MongoDB** - version 4.4 or higher for storing taxonomies and contents
- **Kafka** - version 2.8 or higher

Dependencies

- MongoDB database
- ability to connect to a Kafka instance used by the engine

The service comes with most of the needed configuration properties filled in, but there are a few that need to be set up using some custom environment variables.

Dependencies

MongoDB helm example

Basic MongoDB configuration - helm values.yaml

```
scheduler-mdb:
    existingSecret: {{secretName}}
    mongodbDatabase: {{SchedulerDatabaseName}}
    mongodbUsername: {{SchedulerDatabaseUser}}
    persistence:
      enabled: true
```

```
      mountPath: /bitnami/mongodb
      size: 4Gi
    replicaSet:
      enabled: true
      name: rs0
      pdb:
        enabled: true
        minAvailable:
          arbiter: 1
          secondary: 1
      replicas:
        arbiter: 1
        secondary: 1
      useHostnames: true
    serviceAccount:
      create: false
    usePassword: true
```

DANGER

This service needs to connect to a Mongo database that has replicas, in order to work correctly.

Configuration

Configuring MongoDB

The MongoDB database is used to persist scheduled messages until they are sent back. The following configurations need to be set using environment variables:

- `SPRING_DATA_MONGODB_URI` - the URI for the MongoDB database

Configuring Kafka

The following Kafka related configurations can be set by using environment variables:

- `SPRING_KAFKA_BOOTSTRAP_SERVERS` - address of the Kafka server
- `SPRING_KAFKA_CONSUMER_GROUP_ID` - group of consumers
- `KAFKA_CONSUMER_THREADS` - the number of Kafka consumer threads
- `KAFKA_AUTH_EXCEPTION_RETRY_INTERVAL` - the interval between retries after `AuthorizationException` is thrown by `KafkaConsumer`

Each action available in the service corresponds to a Kafka event. A separate Kafka topic must be configured for each use-case.

> **CAUTION**
>
> Make sure the topics configured for this service don't follow the engine pattern.

Configuring logging

The following environment variables could be set in order to control log levels:

- `LOGGING_LEVEL_ROOT` - root spring boot microservice logs
- `LOGGING_LEVEL_APP` - app level logs

Was this page helpful?

PLATFORM SETUP GUIDES / Data search service setup guide

Introduction

This guide will walk you through the process of setting up the Search Data service using a Docker image.

Infrastructure prerequisites

Before proceeding with the setup, ensure that the following components have been set up:

- **Redis** - version 6.0 or higher
- **Kafka** - version 2.8 or higher
- **Elasticsearch** - version 7.11.0 or higher

Dependencies

- **Kafka** - used for communication with the engine
- **Elasticsearch** - used for indexing and searching data
- **Redis** - used for caching

Configuration

Configuring Kafka

Set the following Kafka-related configurations using environment variables:

- `SPRING_KAFKA_BOOTSTRAP_SERVERS` - address of the Kafka server
- `KAFKA_TOPIC_DATA_SEARCH_IN`
- `KAFKA_TOPIC_DATA_SEARCH_OUT`
- `KAFKA_CONSUMER_THREADS` - the number of Kafka consumer threads

Configuring Elasticsearch

Set the following Elasticsearch-related configurations using environment variables:

- `SPRING_ELASTICSEARCH_REST_URIS`
- `SPRING_ELASTICSEARCH_REST_DISABLESSL`
- `SPRING_ELASTICSEARCH_REST_USERNAME`
- `SPRING_ELASTICSEARCH_REST_PASSWORD`
- `SPRING_ELASTICSEARCH_INDEX_SETTINGS_NAME` - the index can be customized for data-search and it should be similar to what is configured on the process-engine

Configuring authorization & access roles

Set the following environment variables to connect to the identity management platform:

- `SECURITY_OAUTH2_BASE_SERVER_URL`
- `SECURITY_OAUTH2_CLIENT_CLIENT_ID`
- `SECURITY_OAUTH2_REALM`

Configuring logging

The following environment variables could be set in order to control log levels:

- `LOGGING_LEVEL_ROOT` - for root spring boot microservice logs
- `LOGGING_LEVEL_APP` - for app level logs

Elasticsearch

Data search in Elasticsearch runs against an index pattern representing multiple indices. The index pattern is derived from the configuration property:

`spring.elasticsearch.index-settings.name`

Below is an example of a filter to be used in Kibana (as generated by data search):

```
{
  "query": {
    "bool": {
      "adjust_pure_negative": true,
      "boost": 1,
      "must": [
        {
          "nested": {
            "boost": 1,
```

```
            "ignore_unmapped": false,
            "path": "keyIdentifiers",
            "query": {
              "bool": {
                "adjust_pure_negative": true,
                "boost": 1,
                "must": [
                  {
                    "match": {
                      "keyIdentifiers.key.keyword": {

"auto_generate_synonyms_phrase_query": true,
                        "boost": 1,
                        "fuzzy_transpositions": true,
                        "lenient": false,
                        "max_expansions": 50,
                        "operator": "OR",
                        "prefix_length": 0,
                        "query": "astonishingAttribute",
                        "zero_terms_query": "NONE"
                      }
                    }
                  },
                  {
                    "match": {

"keyIdentifiers.originalValue.keyword": {

"auto_generate_synonyms_phrase_query": true,
                        "boost": 1,
                        "fuzzy_transpositions": true,
                        "lenient": false,
                        "max_expansions": 50,
                        "operator": "OR",
                        "prefix_length": 0,
```

```
                        "query": "OriginalGangsta",
                        "zero_terms_query": "NONE"
                      }
                    }
                  }
                ]
              }
            },
            "score_mode": "none"
          }
        },
        {
          "terms": {
            "boost": 1,
            "processDefinitionName.keyword": [
              "TEST_PORCESS_NAME_0",
              "TEST_PORCESS_NAME_1"
            ]
          }
        }
      ]
    }
  }
}
```

Was this page helpful?