

Mon Compte Mobilité

Configuration du SIRH Financeur

# 

# Introduction au document

Ce document décrit les étapes à suivre nécessaires à l’accostage d’un système SIRH pour un financeur ayant choisi ce mode d’intégration, c’est-à-dire que le traitement des souscriptions à une aide est réalisé directement dans l’application SIRH du financeur, par ses gestionnaires/superviseurs.

Les grandes étapes sont :

* Enregistrement d’un client confidentiel dans le fournisseur d’identité de moB
* Configuration du bus de messages de moB

Seuls les paramètres indiqués sont à saisir (attention à la casse).

Les paramètres non listés sont ceux par défaut.

## Client Keycloak

Ce type de client OIDC est plutôt à destination des applications Backend. Il correspond à un compte de service.

Il permet le flux « Authorization Code Flow », le flux « Client Credentials » et de récupérer un jeton de longue durée (si demandé).

A la création, l’équipe MCM renseigne notamment le client ID et génère un client secret.

Une fois créée, le client ID et le client secret sont alors communiqués de façon sécurisée par l’équipe MCM au partenaire SIRH (ex. Neocase Software).

## File d’attentes (queues) Rabbitmq

Les données et changements d’état des souscriptions traitées par SIRH transitent dans des files d’attentes ou queues du bus de messages Rabbitmq. Il faut configurer 2 queues par SIRH financeur.

## Informations requises

On note pour la suite les variables ci-dessous.

|  |  |  |
| --- | --- | --- |
| Nom variable | Description | Exemple |
| *FUNDER\_NAME* | nom du financeur (Entreprise/Collectivité) s’interfaçant avec MOB | capgemini |
| *SIRH\_NAME* | nom de la solution SIRH utilisé par le financeur s’interfaçant avec MOB | neocase |
| *MOB\_SECRET\_KEY* | client secret du client api moB | xxxxxxxxx |
| *SIRH\_SECRET\_KEY* | client secret du client SIRH (obtenu à [cette étape](#_Onglet_Credentials)) | xxxxxxxxx |

# Keycloak

## Client Scopes SIRH

Dans cette section, on va s’intéresser aux client scopes nécessaires à attribuer au client SIRH afin qu’il puisse accéder au bus de message comme prévu dans la spécification.

### Client Scope « Publication de messages de statut »

**Name :** rabbitmq.write.%2F/mob.\*  
**Description :** Permet d’écrire des messages dans l’exchange prévu pour les messages moB afin que ceux-ci soient acheminés vers les files de messages attendues  
**Protocol :** openid-connect  
**Include In Token Scope :** ON

#### Mapper « Audience »

**Name** : Audience

**Mapper type** : Audience

**Included Custom Audience** : rabbitmq

**Add to ID token** : OFF

**Add to access token** : ON

Ce client scope est déjà présent, il n’est pas à recréer.

### Client Scope « Lecture des messages de souscriptions »

**Name :** rabbitmq.read:%2F/mob.subscriptions.put.*FUNDER\_NAME*  
**Description :** Droit de lire et consommer les messages de souscription d’aide présents dans la file d’attente du financeur  
**Protocol :** openid-connect

**Display ON Consent Screen :** OFF  
**Include In Token Scope :** ON

#### Mapper « company »

**Name** : company

**Mapper type** : Hardcoded claim

**Token claim name** : company

**Claim value** : FUNDER\_NAME

**Claim json type** : String

**Add to ID token** : OFF

**Add to access token** : ON

**Add to userinfo** : OFF

### Client Scope « Management (HTTP REST API) »

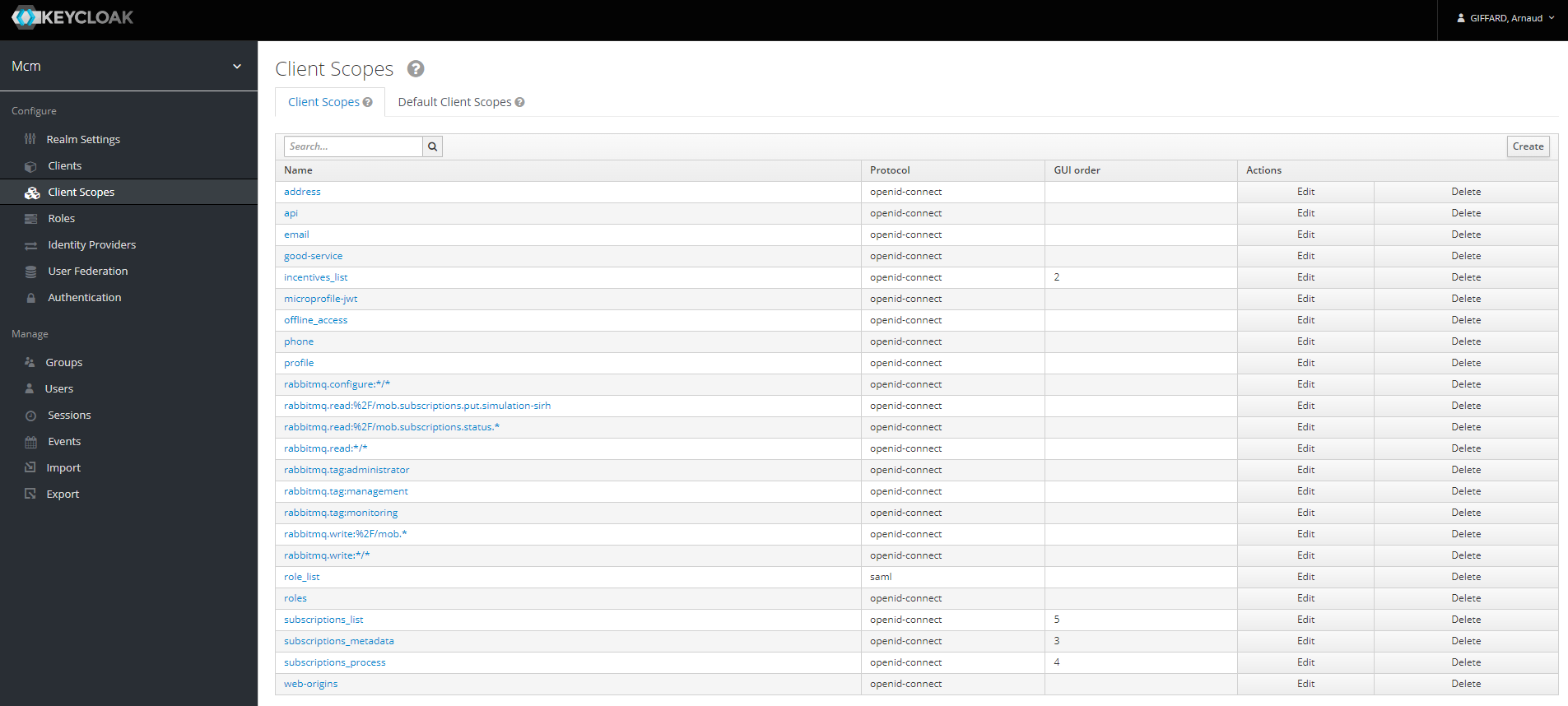
**Name :** rabbitmq.tag:management  
**Description :** Accès l’API HTTP REST du bus de messages  
**Protocol :** openid-connect  
**Include In Token Scope :** ON

Ce client scope est déjà présent, il n’est pas à recréer.

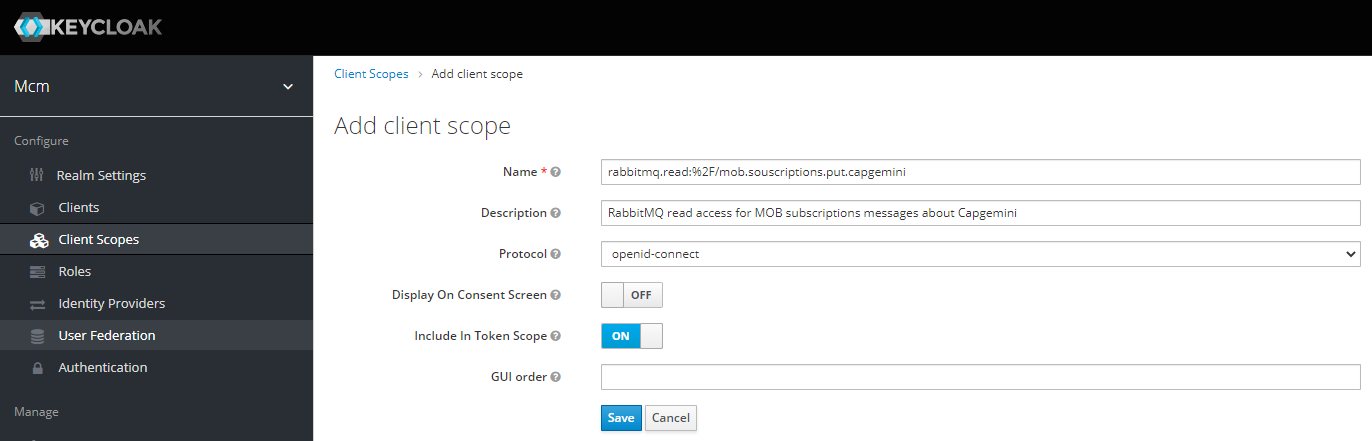
### Etapes de création

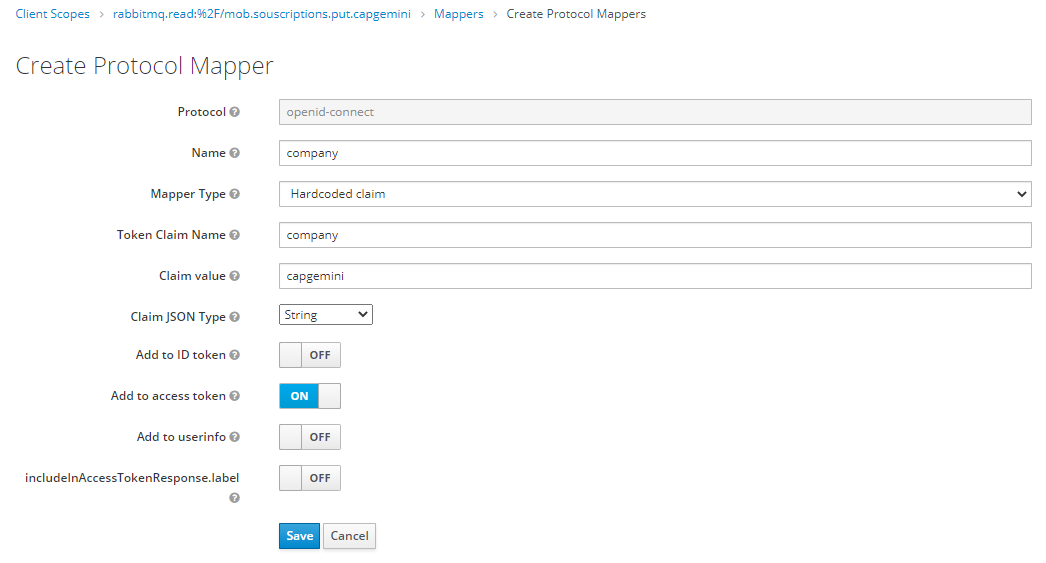
**!!! Les étapes suivantes sont à réaliser une seule fois par environnement, pour chacun des Client Scopes demandés !!!**

1. Aller dans la page Client Scopes via le menu principal puis cliquer sur « Create » pour ajouter un nouveau Client Scope :



1. Complétez les champs par les valeurs requises :





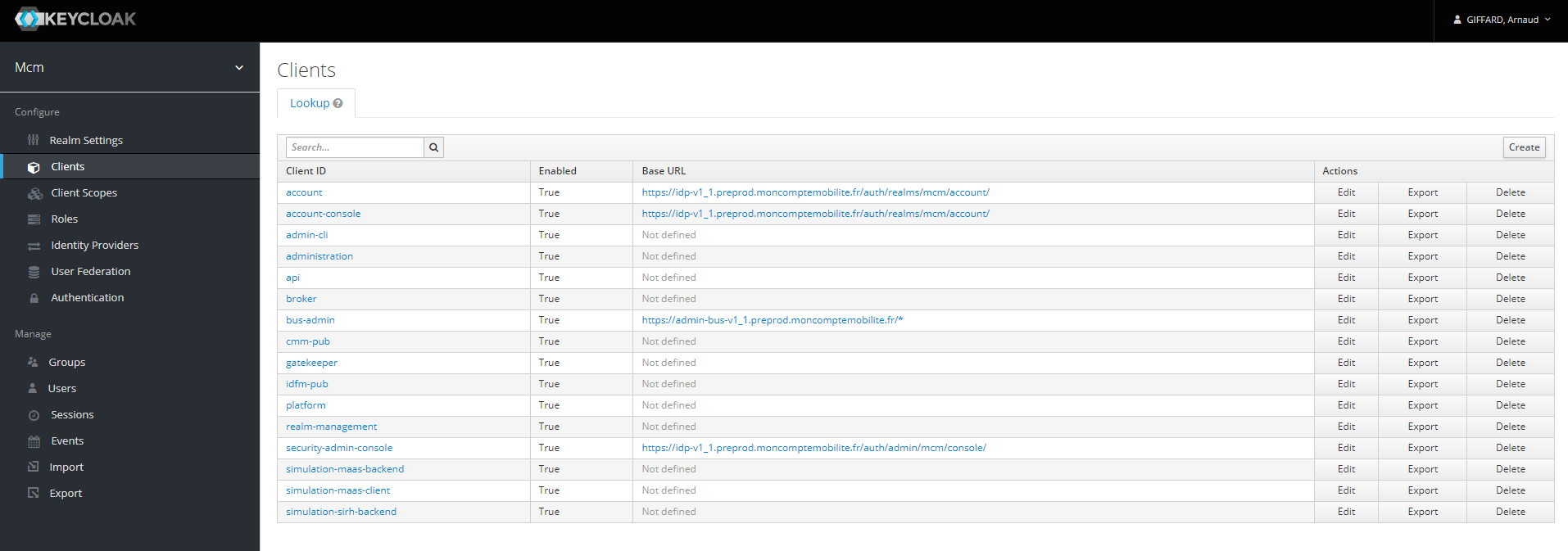
1. Cliquez sur « Save »

## Client SIRH

### Client confidentiel

Dans cette section, on va s’intéresser à la création du client keycloak confidentiel qui sera utilisé par le SIRH pour obtenir un jeton d’accès valider qui lui donnera accès au bus de messages.

Dans l’onglet Clients cliquer sur « Create ».



#### Ecran Add Client

**Client ID** : *FUNDER\_NAME*-sirh (ex. capgemini-sirh)

**Client Protocol :** openid-connect



Cliquer sur Save.

#### Onglet Settings

**Client Id** : *FUNDER\_NAME*-sirh (ex. capgemini-sirh)

**Name** : *FUNDER\_NAME SIRH\_NAME* (ex. Capgemini Neocase)

**Enabled** : ON

**Login Theme** : mcm\_template

**Client protocol** : openid-connect

**Access type** : confidential

**Direct access grants enabled** : OFF

**Service account enabled** : ON

**Valid Redirect UrIs** : \*

Cliquer sur Save.

#### Onglet Credentials

Il faut récupérer le secret et le transmettre au SIRH pour qu’ils puissent se connecter à notre fournisseur d’identité.

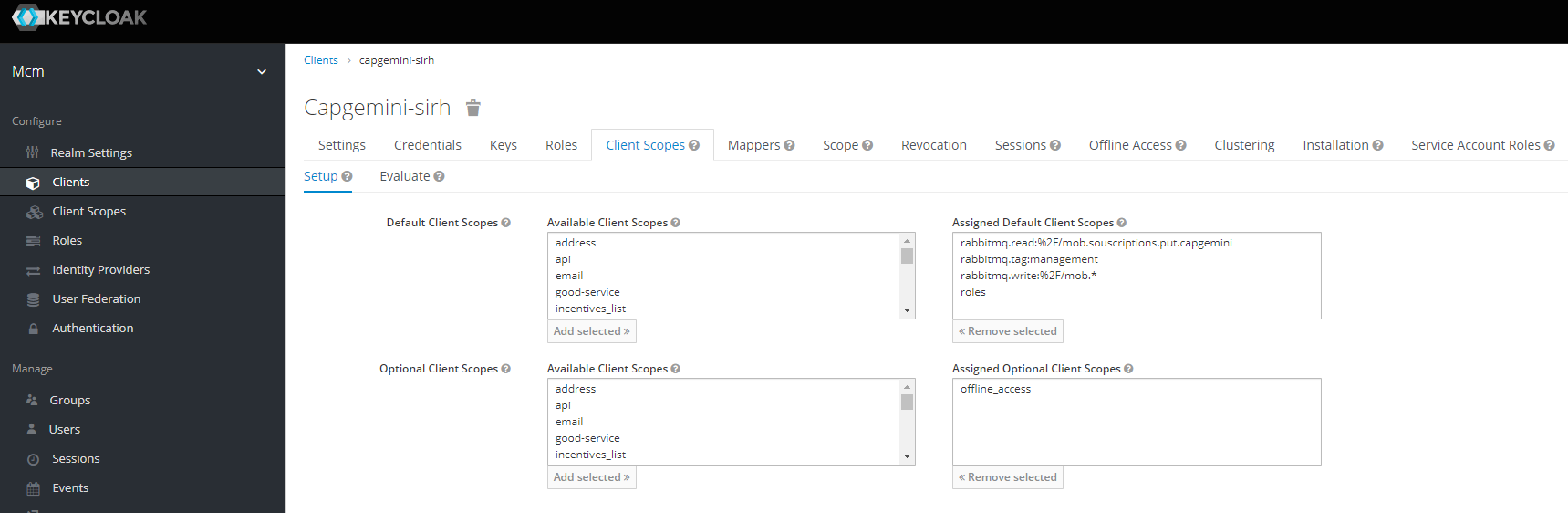
#### Onglet Roles

Dans cet onglet, il faut ajouter un rôle avec le nom *sirh* et un autre avec le nom *service\_sirh* (attention à la casse).

#### Onglet Client Scopes

Dans cet onglet, il faut retirer les rôles non nécessaires. Le client doit avoir les scopes suivants :

* Default client scopes
  + roles
  + rabbitmq.read:%2F/mob.subscriptions.put.FUNDER\_NAME
  + rabbitmq.write:%2F/mob.\*
  + rabbitmq.tag:management **(seulement si le SIRH utilise le mode d’interfaçage API pour le bus)**
* Assigned optional scopes
  + offline\_access
  + funders-clients (pour le rendre accessible dans l’écran de création d’une entreprise, à retirer ensuite)



#### Onglet Mappers

Dans cet onglet, il faut ajouter/modifier les mappers suivants avec la configuration ci-dessous.

##### Client ID

Cliquer sur Edit.

**Name** : Client ID

**Mapper type** : User Session Note

**User Session Note** : clientId

**Token claim name** : client\_id (attention à la casse)

**Claim JSON type** : String

**Add to ID token** : ON

**Add to access token** : ON

**Add to userinfo** : OFF

Cliquer sur Save.

##### sirh\_name

Cliquer sur Create.

**Name** : sirh\_name

**Mapper type** : Hardcoded claim

**Token claim name** : sirh\_name

**Claim value** : *SIRH\_NAME-backend* (doit correspondre au nom du financeur et terminer par le suffixe -backend absolument)

**Claim json type** : String

**Add to ID token** : OFF

**Add to access token** : ON

**Add to userinfo** : OFF

Cliquer sur Save.

##### sirh\_role

Cliquer sur Create.

**Name** : sirh\_role

**Mapper type** : Hardcoded role

**Rôle** : sélectionner le client rôle *FUNDER\_NAME-sirh*, et sélectionner le rôle *service\_sirh*.

Cliquer sur Save.

##### groups

Cliquer sur Create.

**Name** : groups

**Mapper type** : Group Membership

**Token claim name** : membership

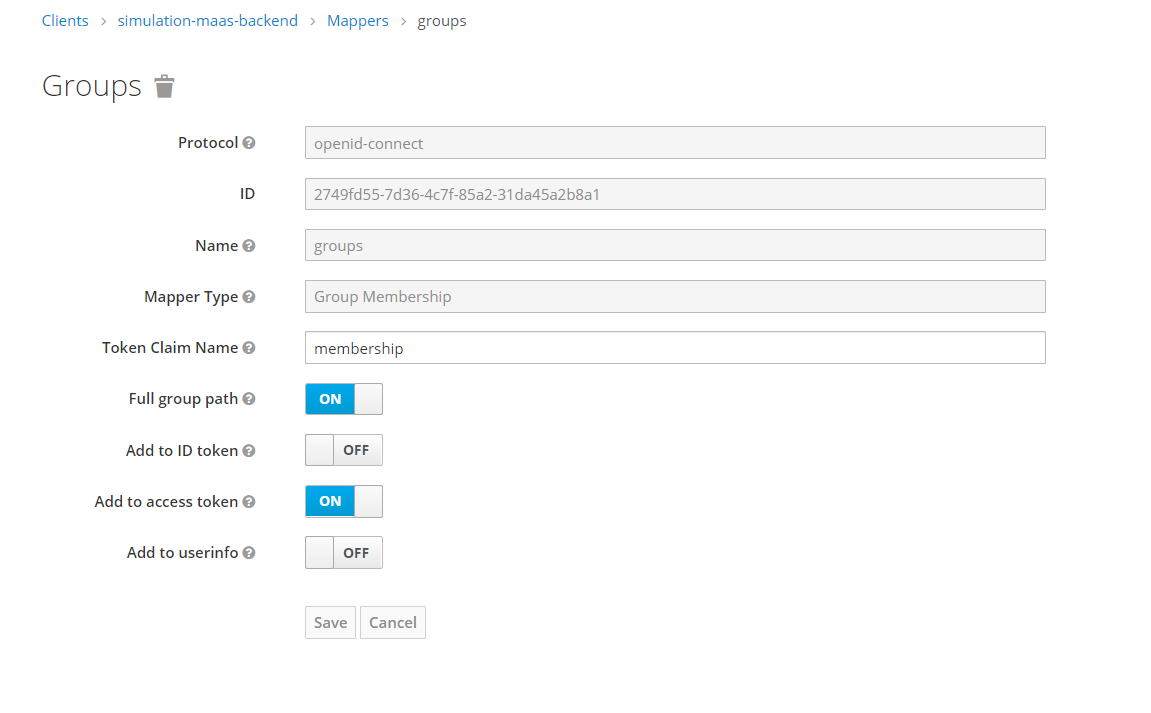
**Full group path** : ON

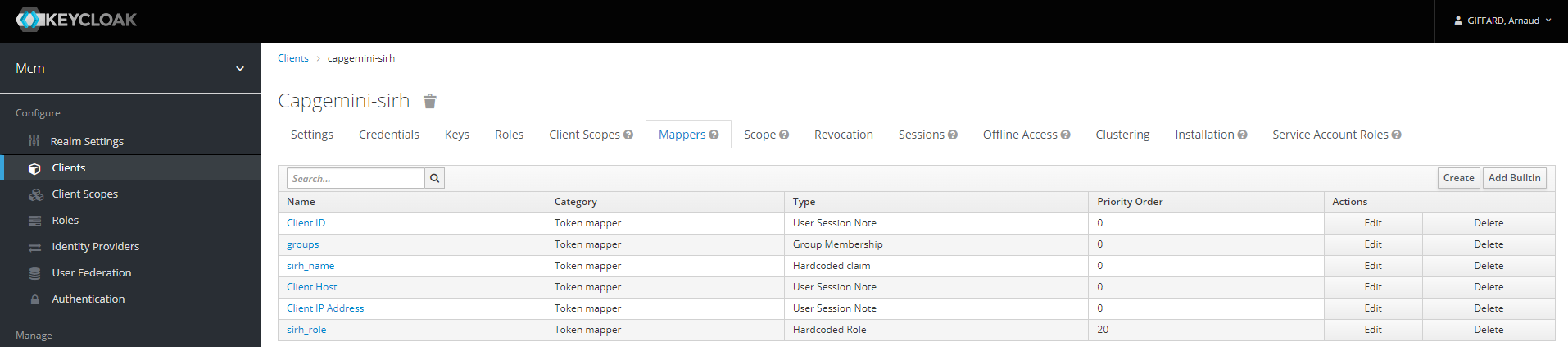
**Add to ID token** : OFF

**Add to access token** : ON

**Add to userinfo** : OFF

Cliquer sur Save.





## Invoquer l’API et l’API BUS

Le fournisseur d’identités permet de donner un JWT, contenant un jeton d’accès.

Pour l’API moB, ce jeton d’accès est un Bearer Token.

Pour l’API BUS, ce jeton d’accès doit être ajouter dans le header, dans le champ password du mode authentification basique. Le champ username peut être vide, le nom d’utilisateur est récupéré à partir du jeton.

Si ce jeton d’accès est expiré, il faudra demander un nouveau jeton d’accès avec les credentials liés au client (client\_id / client\_secret).

# RabbitMQ

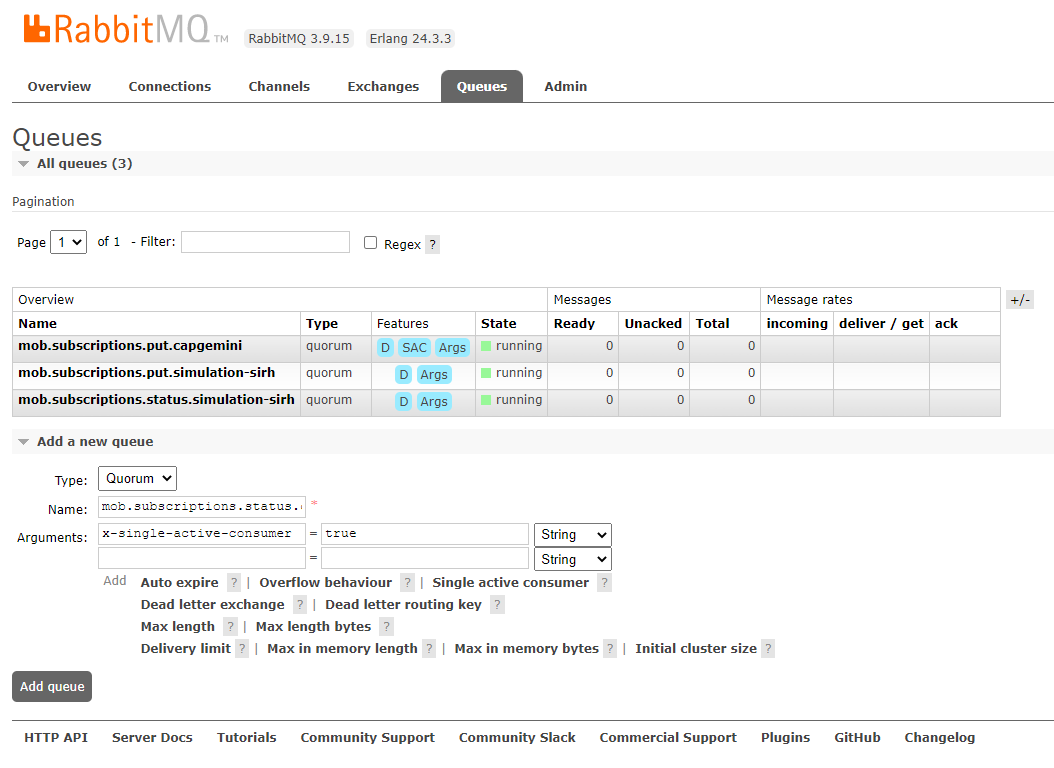
Afin de réaliser les actions, il faut se connecter en tant qu’administrateur sur la console d’administration RabbitMQ.

<https://admin-bus.preprod.moncomptemobilite.fr/>

## Files d’attente (queues)

Dans cette section, on va s’intéresser à la création des queues qui seront utilisés par moB et le SIRH pour s’échanger les informations concernant les souscriptions aux aides du financeur.

Dans l’onglet Queues, déplier le menu « Add a new queue » puis renseigner les informations pour chacune des 2 queues.



### Queue de dépôt

**Type** : Quorum

**Name** : mob.subscriptions.put.*FUNDER\_NAME*

(exemple : mob.subscriptions.put.capgemini)

**Arguments** :

|  |  |  |
| --- | --- | --- |
| x-single-active-consumer | false (si le SIRH utilise l’API http)  true (si le SIRH utilise le protocole AMQP) | Boolean |

Cliquer sur Add queue.

### Queue de statut

**Type** : Quorum

**Name** : mob.subscriptions.status.*FUNDER\_NAME*

(exemple : mob.subscriptions.status.capgemini)

**Arguments** :

|  |  |  |
| --- | --- | --- |
| x-single-active-consumer | true | Boolean |

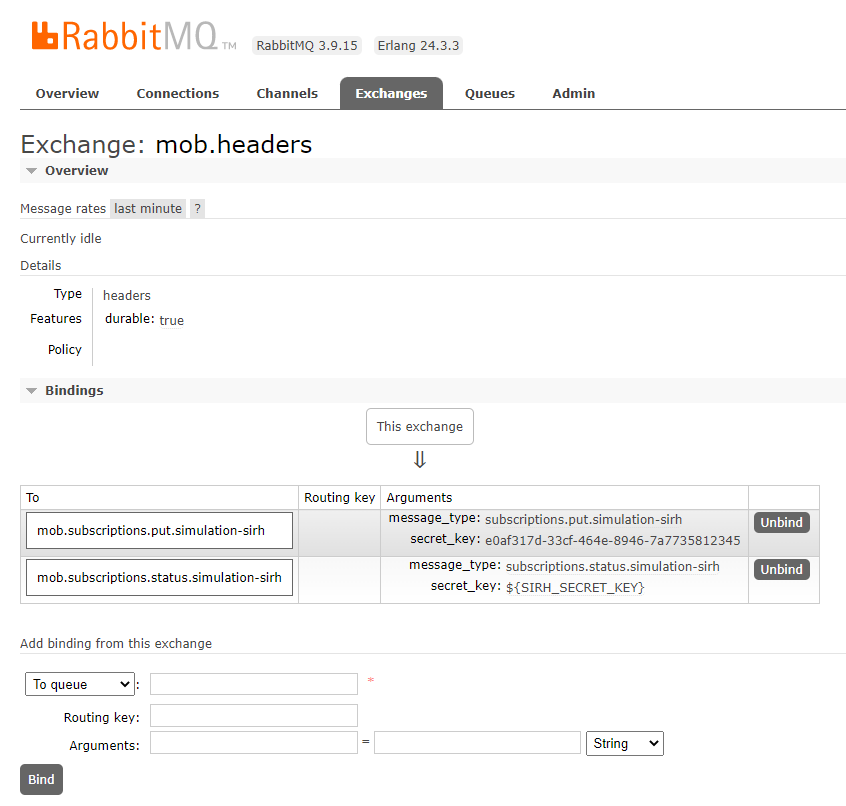
Cliquer sur Add queue.

## Exchanges bindings

Dans cette section, on va s’intéresser à la création des bindings qui vont permettre le routage des messages publiés vers les queues correspondantes.

Dans l’onglet Exchanges, cliquer sur l’exchange « mob.headers ».

Dans la section « Add binding from this exchange » ,renseigner les informations pour chacun des 2 bindings.



### Binding pour la queue de dépôt

**To queue** : mob.subscriptions.put.*FUNDER\_NAME*

(exemple : mob.subscriptions.put.capgemini)

**Arguments** :

|  |  |  |
| --- | --- | --- |
| message\_type | subscriptions.put.*FUNDER\_NAME* | String |
| secret\_key | *MOB\_SECRET\_KEY* | String |

Cliquer sur Bind.

### Binding pour la queue de statut

**To queue** : mob.subscriptions.status.*FUNDER\_NAME*

(exemple : mob.subscriptions.status.capgemini)

**Arguments** :

|  |  |  |
| --- | --- | --- |
| message\_type | subscriptions.status.*FUNDER\_NAME* | String |
| secret\_key | *SIRH\_SECRET\_KEY* | String |

Cliquer sur Bind.

### Activation du consumer sur la queue de statut

Afin que le service api de moB détecte la présence du nouveau SIRH, il faut redémarrer le service api ou attendre le prochain rafraichissement quotidien (2h00).