Service Autosetup Interface

CX Core ART

Exported on 10/27/2022

Table of Contents

Portal Dev.:	**Service Dev.:**

1 Interface / API / Service Summary

For the case of a service setup / app, where an autosetup is available, CX will offer a standardized interface to push the relevant information to the service provider, which can trigger the service setup, if relevant.

Following interfaces are relevant to enable the autosetup

1. POST Service URL
2. POST Service Request
3. POST Instance Details

2 Architecture Overview

2.1 #1 Highlevel Architecture picture

to be added

2.2 #2 Details

to be added

3 Implementation

3.1 #1 POST Service URL

The post service url is necessary since this url storage is holding the service partner / app partner autosetup url.

Logic: the service provider (must be an cx member) can trigger the endpoint to store the autosetup endpoint.

```
{"string"}
```

3.2 #2 POST Service Request

The service request (from the customer, to the provider) is getting stored and managed under this api.

Endpoint: POST /api/services/{serviceId}/subscribe

This API will be used to trigger (if available) the service provider endpoint (stored under #1) with the following body

```
Body:

[
 {
  "customer" :
      {
         "organizationName": "companyName"[required] ,

         "country": "company country alpha2code" [required],

         "email": "user email address which triggered the service" [required]
      }

  "properties":
      {
         "bpnNumber": "bpn of the company" [required],

         "subscriptionId": "" [required],
          "serviceId": "" [required]
        }
      }
]
```

3.3 #3 POST Service Instance

Post service request is used to create the customer service/app instance inside the portal db.

with the successful client/app instance creation on the portal side, the technical user for the AAS registry will get send within the response

Request URL

```
Request
[
 {
   requestId (service request id),
   appUrl (service provider endpoint / client / app)
 }
]
```

Implemented Logic:

i. First execute the service validations
ii. Create a new client id
iii. Create an app instance for the customer (iam_client table and app_instance table)
iv. Create client inside keycloak (with naming convention defined in #2)
v. Add roles into the client (fetched from user_role table for respective app)
vi. Create a technical user for the AAS Registry
vii. update the service/app status in the app_subscription table to "ACTIVE"
viii. create api response message by responding with technical user id and secret
ix. create a customer notification to inform the customer company about the technical user creation via the service provider

Endpoint needed - POST: /api/services/autosetup

Controller: service

<u>Request</u>

- requestId (id in the offer_subscription table)

- appUrl (url for app_instance table)

<u>Response</u>

- technical user id
- technical user secret

API Endpoint Logic

- Validate if requestId is existing in app_subscription table
- Validate if user calling the endpoint is registered as service provider of the service/app
- Validate if the subscription is in status "PENDING"
    - if yes, proceed
    - if no, error
- Run client name creation (logic needed - similar like the service account logic implemented by Norbert; but in this case for the client itself) - ideally naming convention should be something like Cl-{AppName}-{CustomerName}
- Add app instance and client for the customer and app id in following tables
    - iam_client table
    - app_instance table
- Next, create the client in keycloak central IdP - setting
    - Client ID: {client name defined by the service before}
    - Access Type: {public, might get auto set, please check}
    - Standard Flow Enabled: true
    - Direct Access Grants Enabled: true
    - Valid Redirect URIs: {url send via the request body, likely a "*" needs to get added}
    - Web Origins: "+"
    - Backchannel Logout Session Required: true
    - Full Scope allowed: false
- now add roles to the same client by using the roles stored inside user_roles in the portal db and linked to the respective app for which the instance got created
- Create the technical user for the service, by creating another client with following settings
    - Client ID: {ideally client name + prefix "sa-"}
    - Access Type: {confidential}
    - Standard Flow Enabled: false
    - Direct Access Grants Enabled: false
    - Service Accounts Enabled: true
    - Backchannel Logout Session Required: true
    - Full Scope allowed: true
    - Service Account Roles: select the service account role "Digital Twin Management" of the client "technical_roles_management"
- add to the technical user the bpn as attribute (bpn of the customer) and the bpn mapper inside the client. Config see attachment
- store technical user data inside portal db => description "Technical User for app {app name} - {techUserRoleName}"

- Technical user to be mapped to the customer company id
- Back inside the portal db, update the service/app status in the app_subscription table from "PENDING" to "ACTIVE"
- Create Notifications
    - customer notification to inform the customer company about the technical user creation via the service provider. (receiver: customer IT admin)
    - customer notification to inform the customer company about the activated app/service (receiver: customer app requester, customer IT admin)

Permission: "activate_subscription"

Response URL

```
Response

 {
   technicalUserId,
   technical user secret
 }
```

4 Change and Further Development of the Interface

n/a

5 Implementation Levels

n/a

6 Open Tasks:

n/a