Utilisation d'un référentiel externe pour les points de contact GeoNetwork

Travail exploratoire

24 février 2023

Jean Pommier
jean.pommier@pi-geosolutions.fr

Introduction

GeoNetwork offre plusieurs façons de gérer les points de contact : soit à la main dans le formulaire d'une fiche de métadonnées, soit via un annuaire de points de contacts. L'avantage de cette deuxième méthode est similaire à ce qu'on fait avec les thesauri ; en incitant les éditeurs de métadonnées à utiliser un annuaire, on limite le risque de doubles saisies, d'erreurs de frappe, etc. On accélère aussi la saisie des métadonnées.

L'INPN fournit un référentiel organismes, via une API librement accessible. Si celui-ci n'est pas vraiment idéal (cf. infra) pour alimenter l'annuaire de points de contacts de GeoNetwork, il a l'avantage d'exister et de permettre des expérimentations.

Le présent document est un compte-rendu de ces expérimentations et de mes réflexions.

Tout le code a été écrit en Python et est disponible sur https://github.com/pi-geosolutions/contacts_sync.

Analyse de l'API référentiel organismes de l'INPN


```
▾ {
    organizationId: "106",
    address: "181 ALLEE DU CASTEL SAINTE CLAIRE",
    city: "HYERES",
    organizationCode: "5A433BD0-1FFE-25D9-E053-2614A8C026F8",
    actionPerimeterCode: "4",
    typeCode: "5",
    statusCode: "1",
    membershipLevelCode: "1",
    country: "FRANCE",
    formCreationDate: "2018-05-15",
    modificationDate: "2019-11-12",
    email: null,
    frozen: "0",
    shortName: "PN PORT-CROS",
    longName: "PARC NATIONAL DE PORT-CROS",
    postCode: "83406",
    headOfficeSiret: "18830005700109",
    url: http://www.portcros-parcnational.fr/fr,
    description: null,
    siret: "18830005700109",
    geographicalAreaDetail: null,
    subscriptionDate: "1968-12-31",
    phone: "0494128230",
    state: null,
    knowledgeAreaCodes: [ ],
    parentCode: null,
    approvalAdministrationCode: null,
    associatedCodes: [ ],
    roleInpnDescription: null,
    roleSinpCode: [ ],
    openObsContributor: false,
    logoUrl: "logo_PNF_Port-cros.png",
    inpnPartner: true,
    enrichedLabel: "Parc national du Port-Cros",
  ▾ _links: {
    ▾ self: {
        href: http://odata-sinp.mnhn.fr/organizations/5A433BD0-1FFE-25D9-E053-2614A8C026F8
      }
    }
```


FIGURE 1 – Exemple d'entrée fournie par l'api INPN

Cette API couvre nos besoins en termes de tests, mais sera difficile à utiliser avec satisfaction pour le besoin exprimé ici. Je vois deux principaux obstacles :

- La plupart des entrées texte sont tout en CAPITALES. Cela les rend difficilement utilisables dans les entrées de points de contact : c'est désagréable à lire tel quel, et

délicat à convertir en minuscules. On peut certes les convertir en minuscules sauf la première lettre (mais quid des noms propres ?), ou chaque mot commence par une majuscule (mais ça ne présente pas toujours au mieux). Dans tous les cas, on perd les caractères accentués. Il aurait été préférable de stocker les entrées textes dans leur casse naturelle, il est toujours facile *a posteriori* de les passer en majuscule quand le besoin se fait sentir.

- Le manque d'éléments de filtre. La base expose par défaut 7593 entrées, ça fait beaucoup. Il faudrait pouvoir filtrer les entrées et ne retenir que ce qui est vraiment pertinent. Le seul attribut qui me semble potentiellement utilisable est `inpnPartner`, ce qui réduit le nombre d'entrée à 104, mais je ne suis même pas sûr que ce soit pertinent.

Flux de traitement envisagés

J'ai considéré deux principales chaines de traitement :

1. Collecter les entrées de l'API et pousser directement les entrées dans la base de données (court-circuiter GeoNetwork).
2. Collecter les entrées de l'API, écrire les entrées sous forme de fichiers XML et les importer via GeoNetwork.

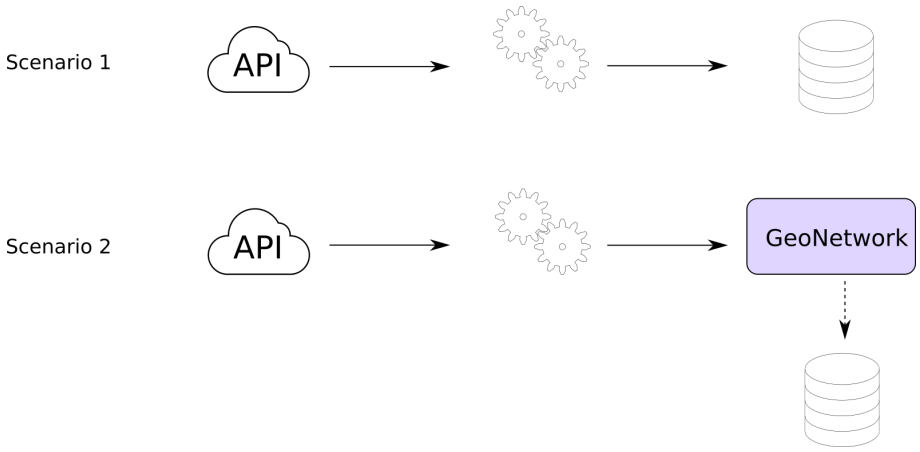

FIGURE 2 – Workflows envisagés

Le scenario 1 semble plus simple, mais est aussi plus fragile car la BD est gérée en principe par GeoNetwork et offre moins de possibilités que l'interface d'import de GeoNetwork.

Le scenario 2 est plus propre, on utilise le flux attendu, mais présente l'inconvénient de passer par authentification gérée par le CAS, un peu plus délicate à utiliser.

J'ai retenu le scenario 2. Le code écrit dans ce contexte pourra resservir sur d'autres besoins d'interactions avec l'API de GeoNetwork.

Le flux de traitement se découpe en deux étapes :

1. Collecter les entrées de l'API, écrire les entrées sous forme de fichier XML
2. importer les fichiers XML dans GeoNetwork.

API INPN -> XML

C'est l'étape la plus simple : pas d'authentification, flux json, on est dans un cas standard. Le code est sur https://github.com/pi-geosolutions/contacts_sync/bl

ob/main/src/contacts_api_to_xml.py.

Le flux est le suivant :

1. On itère dans la pagination des données (gérée par l'API)
2. Pour chaque page, pour chaque entrée dans la page, on récupère l'entrée sous forme de dict (clefs-valeur)
3. et on fournit ce dict, à cahque fois, comme entrée sur un template Jinja2 : les valeurs fournies servent à alimenter un template de fichier XML
4. et on sauvegarde le fichier XML ainsi généré, c'est notre point de contact sous forme de fragment XML conforme ISO 19139

XML -> import dans GeoNetwork

On a plusieurs façons de procéder :

- Via l'interface graphique, fonction d'import : c'est la première que j'ai testé, pour m'assurer que les fichiers XML produits s'importaient correctement.
- Via l'API GeoNetwork, avec pour objectif de ne pas impliquer d'action manuelle. Plus compliqué, il faut gérer l'authentification via le CAS, de façon programmatique et manoeuvrer correctement l'API. C'est l'option que je retiens, bien sûr, puisque le but est de pouvoir automatiser le flux.

Dans les deux cas, on a plusieurs façons d'importer nos points de contact :

- Un par un, on charge les fichiers XML. Bon pour tester, mauvais pour automatiser (long, nombreuses requêtes).
- Créer une archive MEF comme on fait pour les métadonnées, et import en groupe. L'inconvénient, c'est que je n'ai pas les spécifications d'une archive MEF pour ce type d'enregistrements.
- Il est aussi possible d'importer le contenu d'un dossier déposé sur le serveur, accessible par GeoNetwork via son système de fichiers. C'est un peu moins flexible, mais une option intéressante, pas trop compliquée à mettre en oeuvre dans notre configuration (composition docker, possibilité de monter les volumes partagés pour plusieurs conteneurs).

J'ai implémenté les solutions 1 et 3 (au choix via une option de commande). J'ai posé une question sur la mailing-list GeoNetwork pour avoir plus d'infos sur la piste 2 (archive MEF) qui pourrait être propre. Dans la pratique, je pense que la solution 3 est la meilleure pour nous.

Exemple de fichier importé dans GeoNetwork

```
<gmd:CI_ResponsibleParty xmlns:gmd="http://www.isotc211.org/2005/gmd"
  xmlns:gco="http://www.isotc211.org/2005/gco"
  uuid="inpn-contacts-ref-5A433BD0-1FBE-25D9-E053-2614A8C026F8">
  <gmd:individualName>
      <gco:CharacterString/>
   </gmd:individualName>
  <gmd:organisationName>
      <gco:CharacterString>Biotope</gco:CharacterString>
   </gmd:organisationName>
  <gmd:positionName>
      <gco:CharacterString/>
   </gmd:positionName>
  <gmd:contactInfo>
      <gmd:CI_Contact>
        <gmd:phone>
          <gmd:CI_Telephone>
            <gmd:voice>
              <gco:CharacterString>0467184620</gco:CharacterString>
```

```
            </gmd:voice>
            <gmd:facsimile>
               <gco:CharacterString/>
            </gmd:facsimile>
         </gmd:CI_Telephone>
      </gmd:phone>
      <gmd:address>
         <gmd:CI_Address>
            <gmd:deliveryPoint>
               <gco:CharacterString>22 boulevard marechal foch</gco:CharacterString>
            </gmd:deliveryPoint>
            <gmd:city>
               <gco:CharacterString>Meze</gco:CharacterString>
            </gmd:city>
            <gmd:administrativeArea>
               <gco:CharacterString/>
            </gmd:administrativeArea>
            <gmd:postalCode>
               <gco:CharacterString/>
            </gmd:postalCode>
            <gmd:country>
               <gco:CharacterString>France</gco:CharacterString>
            </gmd:country>
            <gmd:electronicMailAddress>
               <gco:CharacterString>siegesocial@biotope.fr</gco:CharacterString>
            </gmd:electronicMailAddress>
         </gmd:CI_Address>
      </gmd:address>
      <gmd:onlineResource>
         <gmd:CI_OnlineResource>
            <gmd:linkage>
               <gmd:URL>http://www.biotope.fr</gmd:URL>
            </gmd:linkage>
            <gmd:protocol>
               <gco:CharacterString>WWW:LINK-1.0-http--link</gco:CharacterString>
            </gmd:protocol>
            <gmd:name>
               <gco:CharacterString/>
            </gmd:name>
            <gmd:description>
               <gco:CharacterString/>
            </gmd:description>
         </gmd:CI_OnlineResource>
      </gmd:onlineResource>
   </gmd:CI_Contact>
</gmd:contactInfo>
<gmd:role>
   <gmd:CI_RoleCode codeList="./resources/codeList.xml#CI_RoleCode" codeListValue="pointOfContact"/>
</gmd:role>
</gmd:CI_ResponsibleParty>
```

Complications

CAS

L'authentification via le CAS m'a posé des soucis, car c'est un cas de figure modérément documenté.

Au final, la solution est assez étonnante de simplicité. Le prototype pour cette étape ne fait que quelques dizaines de lignes de code, aérées. La logique d'authentification fait une dizaine de lignes. On peut voir le résultat sur https://github.com/pi-geosolutions/contacts_sync/blob/main/src/contacts_api_to_apiv2.py.

API GeoNetwork

Je n'ai pas trouvé l'API de GeoNetwork très facile à utiliser, malgré la mise à disposition d'une interface swagger, il m'a fallu un peu de temps pour m'y retrouver. Au final, l'endpoint d'import fonctionne bien, c'est en fait celui qui est utilisé par l'interface graphique, donc le fonctionnement est le même.

Script de la *proof of concept*

Le script "final" est https://github.com/pi-geosolutions/contacts_sync/blob/main/src/contacts_api_to_apiv3.py. Il est un peu plus long, mais c'est essentiellement la config du package click pour le rendre modulaire (options de script, possibilité de passer ces options via des variables d'environnement).

Il reprend, dans son essence, les fonctions des deux autres scripts et propose au choix, soit de les lancer individuellement (pratique pour décortiquer et débugger) soir de chainer les deux traitements (option inpn-to-gn).

Mise en oeuvre en production

Pour une mise en oeuvre en production, je préconise la méthode suivante :

- Création d'un dossier upload dans le volume geonetwork_data
- Ajout d'un service dans la composition docker, avec une image docker contenant ce script (installation avec setuptools, idéalement)
  - ⋆ Le service resterait inactif mais tournerait en tâche de fond.
  - ⋆ Montage du dossier `geonetwork_data/upload` comme volume
  - ⋆ Exécution du script via cron :

```
docker-compose exec contacts_sync contacts_api_to_api inpn-to
    ↪ -gn les_options_du_script
```

Pistes d'améliorations

Parmi les améliorations possibles :

- Permettre une mise à jour sélective des points de contact. l'API INPN fournissant un uuid pour chaque entrée, on peut détecter si une entrée dans l'annuaire de GeoNetwork vient du référentiel INPN. On peut donc à la prochaine mise à jour récupérer les deux sources, et mettre à jour le point de contact de façon sélective : n'actualiser que le nom et l'adresse par exemple, mais pas l'email ni le téléphone. Un traitement de ce genre sera nettement plus lent, puisqu'il faudra requêter l'API GeoNetwork pour récupérer chacune des entrées à concilier. Mais il n'y a pas de difficulté technique majeure.
- Finalisation du script. Il s'agit là d'une *proof of concept*, mais il manque pas mal d'éléments pour en faire un script propre :
  - ⋆ Robustifier le script : gestion des erreurs possibles (réseau, API, etc)
  - ⋆ Logging (pour l'instant on fait juste du printf)
  - ⋆ Rapportage / métriques pour le suivi du fonctionnement du script. On pourrait assez facilement écrire un fichier de métriques Prometheus (ou autre) pour suivi des métriques et alerting.

Conclusion

Grâce à l'API de GeoNetwork, on voit qu'on peut envisager l'utilisation d'un référentiel de point de contact externe, avec une synchronisation régulière. On pourrait même envisager une mise à jour sélective de ces points de contacts, ce qui permettrait de récupérer des informations de base depuis le référentiel externe (nom de l'organisme, adresse) et de les compléter dans GeoNetwork (email, téléphone éventuellement, nom de personne).

L'essentiel du code écrit dans ce script est de l'enrobage (syntaxe click pour fonctions de CLI) mais la base de code pour faire cette synchronisation est pour l'instant très réduite. La mise à jour sélective augmenterait le volume de code, mais on resterait dans un cadre très raisonnable, assez facile à maintenir.