Documentation du script de téléchargement des produits Sentinel-1 et Sentinel-2

Nicolas Débonnaire*

April 4, 2016

Résumé

Le présent script permet de télécharger, grâce à l'API du scihub, les produits Sentinel-1 et Sentinel-2 publiés sur https://scihub.copernicus.eu/ et de détecter automatiquement les nouveaux produits publiés régulièrement afin de les télécharger.

1 Présentation du script

1.1 Principe général du script

Ce script permet de télécharger automatiquement les produits publiés sur le scihub (via l'API) correspondant à une requête OpenSearch donnée. Étant destiné à être lancé de manière régulière (manuellement ou grâce à la programmation d'une tâche), l'archive scihub est scanné afin de détecter ou non la venue de nouveau produits et de les télécharger le cas échéant. Les serveurs étant peu stables ou soumis à des maintenances régulières, une vérification de l'intégrité

*e-mail: ndebonnaire@unistra.fr

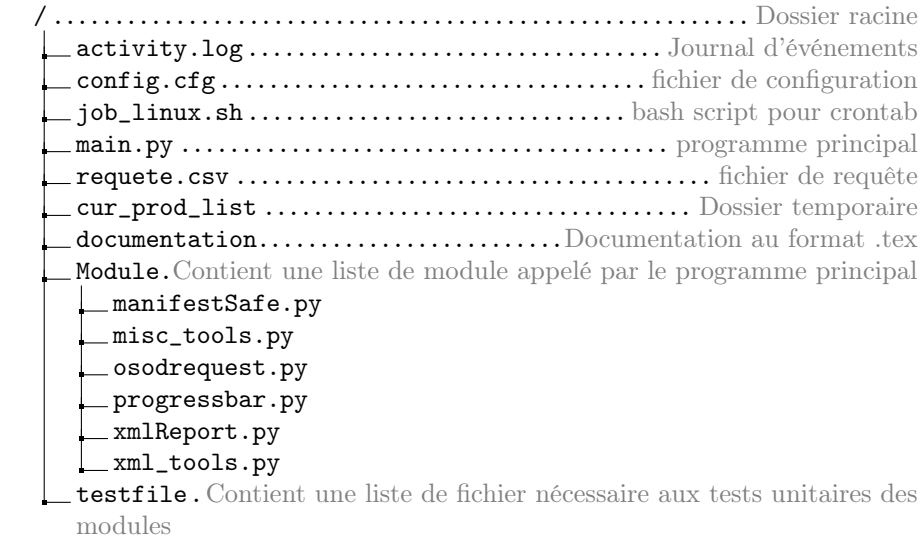

FIGURE 1 – Liste des fichiers principaux.

des produits téléchargés, à partir du checksum md5, est systématiquement réalisée. Enfin, les produits étant relativement coûteux en terme d'espace disque, il est possible de spécifier, pour les produits S2 uniquement, les tuiles et les bandes souhaitées afin de s'affranchir du téléchargement de l'archive complète.

1.2 Organisation des fichiers

La figure 1 présente l'ensemble des fichiers et dossiers présent sous le dossier racine après un premier lancement du script. Seul les fichiers principaux sont indiqués et certains dossiers secondaires (car_prod_list et testfile) ne sont pas détaillés.

Le programme, écrit en python, est organisé en un programme principal *main.py* faisant appel à un ensemble de fonctions repartit dans des modules de manière thématique dans le dossier *Module*. De manière rapide, le module *manifestSafe.py* permet de lire et traiter le fichier de description manifest.safe d'un produit. Le module *misc_ tools.py* contient un ensemble de petites fonctions diverses. Les module *osodrequest.py* et *progressbar* regroupent toutes les fonctions reposant sur les requêtes client/serveur du protocole OpenSearch et Opendata. Le module *xmlReport.py* contient des fonctions permettant de lire et générer des fichiers de rapport de téléchargement au format xml. Ces fichiers rapport contiennent des listes de produits avec, entre autres, une balise 'statut' indiquant l'intégrité du téléchargement. Enfin, le module *xml_ tools.py* contient quelques fonctions permettant d'extraire des informations d'un fichier xml renvoyé par le serveur dans le cadre d'une requête Opensearch.

```
[log]
user = scihub_usermame
pw = scihub_password
auth_url = https://scihub.copernicus.eu/dhus

[param]
dl_dir = /home/user/sentinel_product
nb_retry = 0
wait_time = 240
max_items = 500
```

FIGURE 2 – Structure du fichier de configuration.

2 Comment ça marche ?

Dans cette section, les 3 fichiers principaux nécessaires à l'exécution du script (2 fichiers texte 'config.cfg' et 'request.csv' nécessaire d'être remplis au préalable, et le programme principal de téléchargement des produits 'main.py') sont expliqués.

2.1 Le fichier de configuration, 'config.cfg'

La figure 2 montre la structure du fichier de configuration. La première section [log] contient les paramètres suivant :

- ***user*** : Nom d'utilisateur sur le site du scihub
- ***pw*** : Mot de passe sur le site du scihub
- ***auth_ url*** : Url du scihub ou l'on s'authentifie

Afin de pouvoir accéder à l'API, il est nécessaire de posséder un compte crée avant le 22 mars 2016 [1]. Cette date est susceptible de changer comme par le passé.

La seconde section [param] contient les paramètres suivants :

- ***dl_ dir*** : Chemin du dossier de stockage des produits
- ***nb_ retry*** : Nombre de tentative de connexion au serveur avant d'abandonner et de passer au téléchargement suivant.
- ***wait_ time*** : Temps d'attente, en secondes, entre chaque tentative. Scihub recommande d'attendre 5 minutes avant de réessayer un téléchargement lorsque le site est en maintenance par exemple [2].
- ***max_ items*** : Nombre maximum d'item par requête Opensearch. Actuellement, ce nombre ne semble pas avoir de limite, mais peut être qu'un jour le nombre de produit retourné par une requête sera limité. Par exemple si le nombre total de produit pour une requête donnée et à un instant donné est de 1200 et que la valeur de *max_ items* est 500 alors il faudra parcourir l'archive 3 fois pour récupérer tout les produits. En revanche si la valeur de ce paramètre est à 2000 alors une seule requête est nécessaire.

1. https://scihub.copernicus.eu/news/News00058
2. https://scihub.copernicus.eu/news/News00040

2.2 Le fichier de requêtes, 'request.csv'

La figure 3 montre la structure du fichier de requête. Ce fichier est au format csv et utilise comme séparateur le ';'. Le premier champ **zone_etude** correspond au nom du dossier où seront enregistré les produits correspondants à la requête spécifiée. Le champ **requete** correspond à la requête Opensearch. Seul les paramètres de la requête doivent être spécifié. Par exemple pour la requête `https://scihub.copernicus.eu/dhus/search?q=platformname:Sentinel-2 AND footprint:"Intersects(POLYGON((6.55 44.31, 6.55 44.47, 6.86 44.47, 6.86 44.31, 6.55 44.31)))"` seul *platformname :Sentinel-2 AND footprint :"Intersects(POLYGON((6.55 44.31, 6.55 44.47, 6.86 44.47, 6.86 44.31, 6.55 44.31)))"* doit être écrit. De plus il est impératif de spécifier le nom du satellite avec soit *platformname :Sentinel-2* ou *plateformename :Sentinel-1*[3] pour rechercher les produits Sentinel-2 ou Sentinel-1 respectivement, auquel cas la requête sera considérée comme non valide par le script. Il est également important de ne pas ajouter les paramètres *rows* et *start*[4] qui sont des paramètres permettant de parcourir l'archive. En effet, ces paramètres sont automatiquement ajoutés par le script lors de la lecture de l'archive produit. Pour plus d'information concernant l'écriture et la syntaxe des requêtes OpenSearch veuillez vous référer à la page suivante : `https://scihub.copernicus.eu/twiki/do/view/SciHubUserGuide/3FullTextSearch`. Les 3 derniers champs sont uniquement valable dans le cas d'une requête S2 (c'est-à-dire contenant platformname :Sentinel-2). Le paramètre **nuage** permet de d'indiquer la valeur maximale acceptable du pourcentage de couverture nuageuse. Le champ **Tiles** permet de spécifier les tuiles du produit que l'on veut récupérer et le champ **bands** les bandes souhaitées. Pour ces deux derniers champs, il est possible de spécifier une ou plusieurs tuiles et/ou une ou plusieurs bandes séparées par des virgules (avec ou sans espace après les virgules, peu importe). Il est également possible de ne spécifier que les bandes désirées et de laisser le champ tuile vide auquel cas toute les tuiles seront récupérer avec seulement les bandes indiquées et inversement. Si aucune tuile et/ou bande n'est trouvé, alors le produit ne sera pas téléchargé. Enfin, si ces 2 champs sont laissés vide alors l'archive complète sera téléchargé. (Pour les produits Sentinel-1, la version actuel du script ne permet que de récupérer l'archive entière.) Les produits Sentinel-2 sont découpés selon le système militaire MGRS (Military Grid Reference System) dont des informations détaillées et des couches vecteurs au format shapefile peuvent être trouvées sur le lien suivant : `http://earth-info.nga.mil/GandG/coordsys/grids/universal_grid_system.html#zza4`. De plus une carte interactive de ce système est consultable sur le lien suivant : `http://www.mappingsupport.com/p/gmap4.php?tilt=off&mgrs=14SPG34308382&z=5&t=t1`.

3. https://scihub.copernicus.eu/twiki/do/view/SciHubWebPortal/APIHubDescription
4. https://scihub.copernicus.eu/twiki/do/view/SciHubUserGuide/5APIsAndBatchScripting#Discover_the_list_of_the_product

2.3 Le programme principal, 'main.py'

L'algorithme 1 montre le déroulement du programme principal de manière schématique. Chaque requête (c'est-à-dire chaque ligne) du fichier *'request.csv'* est traitée une à une (boucle for externe). L'algorithme peut être décomposé en 2 étapes. Une première étape (ligne 2 à 18 de l'algorithme) consiste à télécharger à nouveau les produits précédemment téléchargé potentiellement corrompu. La seconde étape (ligne 19 à 39) va vérifier la présence de nouveau produit dans l'archive puis les télécharger le cas échéant.

3 Organisation du dossier de téléchargement

La figure 4 montre de manière schématique la structure du répertoire où seront téléchargé les produits. Le dossier racine *dl_dir* représente le dossier indiqué dans le fichier de configuration (voir figure 2). Les produits sont ensuite classés par zone d'étude (premier champs du fichier de configuration), par satellite (S1 ou S2) et par année (année du début d'acquisition de la scène).

Dans chaque dossier 'zone d'étude', se trouve 2 rapports globaux au format xml, un pour Sentinel-1 et un autre pour Sentinel-2. Ces rapports synthétisent l'ensemble des produits téléchargés et notamment leur intégrité. La figure 5 montre l'exemple d'un rapport pour Sentinel-2. A chaque balise *<entry>* correspond un produit. Chaque produit est décrit par 6 balises enfant :

- ***<title>*** : Le nom du produit,
- ***<id>*** : L'uuid (Universally Unique Identifier) du produit,
- ***<link>*** : Le lien de téléchargement de l'archive complète,
- ***<checksum>*** : Le checksum de l'archive complète du produit,
- ***<status>*** : Le status du téléchargement du produit permettant de savoir si le produit est intègre ou corrompu et,
- ***<year>*** : L'année de début d'acquisition du produit.

Dans le cas de téléchargement d'archive complète (S1, ou S2 avec les champs Tiles et bands vide) la valeur de la balise *<checksum>* peut être soit :

- **Le checksum** (ex :D0AB11311B866B2332E92479906F2FF9) ou,
- **vide** si le checksum n'a pas pu être récupéré pour diverse raison.

De plus, toujours dans le cas de téléchargement de l'archive complète, la valeur de la balise *<status>* peut être soit :

- **'checksum ok'** si le fichier est intègre,
- **'corrupted archive'**, si le fichier est corrompu, ou
- **'missing checksum'**, si le checksum n'a pas pu être récupéré (et donc l'image non téléchargé).

Dans le cas du téléchargement de parties d'archive (S2 avec les champs Tiles et/ou bands non vide) les éléments d'un produit sont téléchargés un à un. Ces

Algorithm 1 Programme principal

```
1: for each Ligne du fichier requête do
2:     Lire le rapport global (voir fig. 5) et extraire puis stocker les produits corrompus dans une liste.
3:     for each Produit corrompu do
4:         if Téléchargement des archives complètes then
5:             Récupérer le lien de téléchargement du produit
6:             Télécharger l'archive
7:         else                                  ▷ Téléchargement des éléments un à un
8:             Lire le rapport détaillé du produit (voir fig. 6) et extraire puis stocker les éléments corrompus dans une liste.
9:             for each Élément corrompus do
10:                Récupérer le lien de téléchargement de l'élément
11:                Télécharger l'élément
12:                Vérifier l'intégrité de l'élément
13:                Mettre à jour le status de l'élément dans le rapport détaillé
14:            end for
15:        end if
16:        Vérifier l'intégrité du produit
17:        Mettre à jour le status du produit dans le rapport global
18:    end for
19:    Récupérer l'ensemble des produits présent dans la base de donnée du scihub
20:    Comparer l'uuid de ces produits avec l'uuid des produits déjà téléchargés afin d'extraire les nouveaux produits publiés
21:    for each Nouveau produits do
22:        if Téléchargement des archives complètes then
23:            Récupérer le lien de téléchargement du produit
24:            Télécharger l'archive
25:        else                                  ▷ Téléchargement des éléments un à un
26:            Télécharger le manifest.safe du produit.
27:            Lire le manifest.safe et extraire la liste des éléments ainsi que leur checksum
28:            Filtrer la liste en fonction des bandes et/ou tuile spécifiées par l'utilisateur dans le fichier request.csv.
29:            Générer les liens de téléchargement pour chaque élément
30:            for each Élément do
31:                Récupérer le lien de téléchargement de l'élément
32:                Télécharger l'élément
33:                Vérifier l'intégrité de l'élément
34:                Ajouter une entrée dans le rapport détaillé
35:            end for
36:        end if
37:        Vérifier l'intégrité du produit
38:        Ajouter une entrée dans le rapport global.
39:    end for
40: end for
```

```
zone_etude;requete;nuage;Tiles;bands
zone1;requeteS1;;;
zone1;requeteS2;20;T1,T2;B1,B2
zone2;requeteS1;;;
zone2;requeteS2;55;T5,T6;B3,B4
```

FIGURE 3 – structure du fichier de requête

éléments sont listés dans le fichier manifest.safe [5] qui doit donc être téléchargé au préalable. Il contient notamment, le checksum de chaque élément du produit sous la balise *<dataObjectSection>*. Dans ce cas, la valeur de la balise *<checksum>* ne peut prendre qu'une seule valeur soit 'not needed' puisque le checksum de l'archive entière n'est pas utilisé pour vérifier l'intégrité de l'ensemble de l'archive. De plus, la valeur de la balise *<status>* peut être soit :

- **'checksum ok'** si l'ensemble des éléments est intègre,
- **'corrupted archive'**, si au moins un des éléments est corrompu, ou
- **'missing manifest'**, si le manifest.safe n'a pas pu être récupéré.

Afin de connaître l'état de chaque élément téléchargé (toujours dans le cas du téléchargement de parties d'archive), un rapport détaillé au format xml est généré pour chaque produit. La figure 6 montre un extrait de rapport détaillé d'un produit. Ce fichier est similaire au rapport global. Chaque balise <entry> correspond cette fois ci à un élément unitaire du produit et chaque élément est décrit par 3 balises enfant :

- ***<path>*** : Le chemin relatif de l'élément dans l'arborescence des dossiers du produit (extrait du manifest.safe),
- ***<link>*** : le lien de téléchargement de l'élément généré à partir du chemin de l'élement,
- ***<checksum>*** : le checksum de l'élément (extrait du manifest.safe) et,
- ***<status>*** : le status du téléchargement de l'élément.

La balise *<checksum>* ne peut prendre qu'une seule valeur soit le checksum de l'élément alors que la balise *status* peut prendre 2 valeurs soit :

- **'checksum ok'** si l'élément est intègre, ou
- **'corrupted file'**, si l'élément est corrompu.

4 Comment l'utiliser ?

4.1 Configuration requise

Ce script à été développé avec la version 2.7 de python et n'a pas été testé avec la version 3.4. De plus, (à cause de l'utilisation d'alarmes (librairie signal)

5. https://scihub.copernicus.eu/twiki/do/view/SciHubUserGuide/5APIsAndBatchScripting#Download_manifest_file_from_the

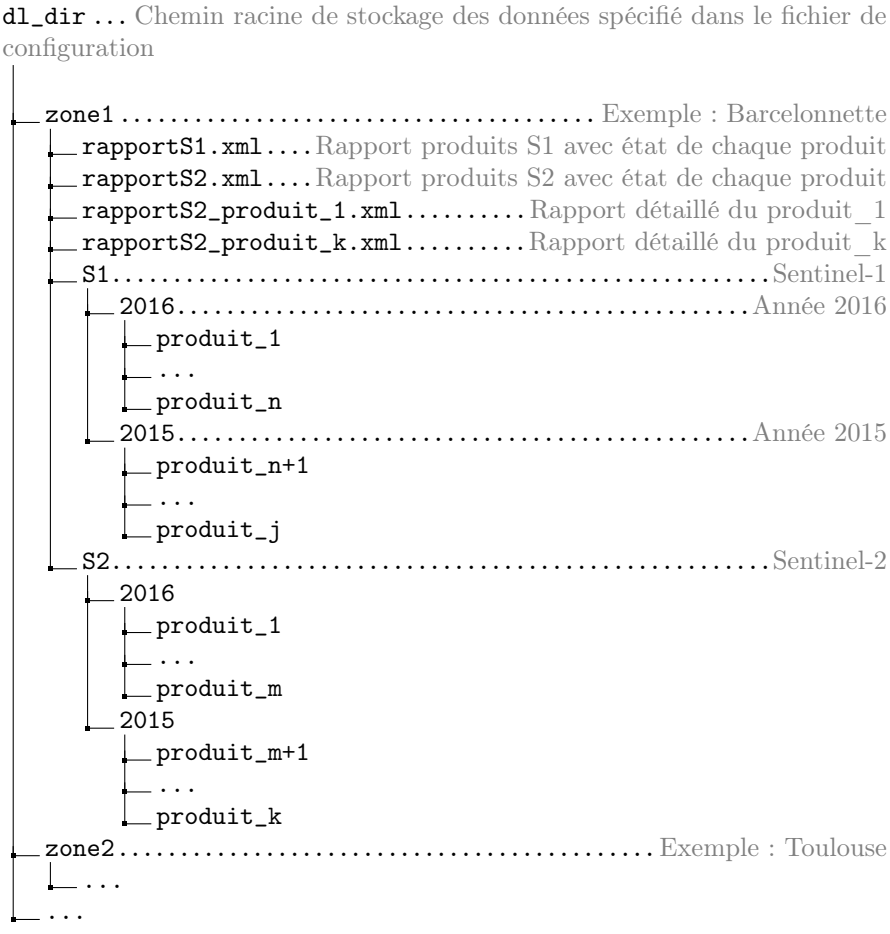

FIGURE 4 – Arbre du dossier de téléchargement

```
<feed>
  <number_past_product>30</number_past_product>
  <entry>
    <title>S2A_OPER_PRD_MSIL1C_PDMC_20160319T232514_R008_
        V20160319T104032_20160319T104032</title>
    <id>42b196da-1ad2-43b8-9359-a7af62938e7c</id>
    <link>https://scihub.copernicus.eu/dhus/odata/v1/Products('42
        b196da-1ad2-43b8-9359-a7af62938e7c')/$value</link>
    <checksum>not needed</checksum>
    <status>checksum ok</status>
    <year>2016</year>
  </entry>
  <entry>
    <title>S2A_OPER_PRD_MSIL1C_PDMC_20160315T145250_R065_
        V20150826T102655_20150826T102655</title>
    <id>1d1c9782-10b7-4eeb-ba74-e1416649d597</id>
    <link>https://scihub.copernicus.eu/dhus/odata/v1/Products('1
        d1c9782-10b7-4eeb-ba74-e1416649d597')/$value</link>
    <checksum>not needed</checksum>
    <status>checksum ok</status>
    <year>2015</year>
  </entry>
  <entry>
    <title>S2A_OPER_PRD_MSIL1C_PDMC_20160113T193242_R065_
        V20160113T103004_20160113T103004</title>
    <id>8e80d69d-0ccb-419c-b120-b0f61b53a1c4</id>
    <link>https://scihub.copernicus.eu/dhus/odata/v1/Products('8
        e80d69d-0ccb-419c-b120-b0f61b53a1c4')/$value</link>
    <checksum>not needed</checksum>
    <status>checksum ok</status>
    <year>2016</year>
  </entry>
</feed>
```

FIGURE 5 – Rapport xml global pour l'ensemble des produits d'une requête et d'un satellite donné. Par souci de lisibilité, le rapport ne montre que 3 produits.

```
<feed>
  <entry>
    <path>./GRANULE/S2A_OPER_MSI_L1C_TL_SGS__20151227T161254_
        A002679_T31TFK_N02.01/QI_DATA/S2A_OPER_MSK_DEFECT_SGS_
        _20151227T161254_A002679_T31TFK_B01_MSIL1C.gml</path>
    <link>https://scihub.copernicus.eu/dhus/odata/v1/Products('
        b28dd648-55ef-45fd-951f-d8259b745b67')/Nodes('S2A_OPER_PR
        D_MSIL1C_PDMC_20151228T100614_R108_V20151227T104738
        _20151227T104738.SAFE')/Nodes('GRANULE')/Nodes('S2A_OPER_
        MSI_L1C_TL_SGS__20151227T161254_A002679_T31TFK_N02.01')/N
        odes('QI_DATA')/Nodes('S2A_OPER_MSK_DEFECT_SGS__20151227
        T161254_A002679_T31TFK_B01_MSIL1C.gml')/$value</link>
    <checksum>176847d558a2be3648d5b9309a03b46e</checksum>
    <status>checksum ok</status>
  </entry>
  <entry>
    <path>./GRANULE/S2A_OPER_MSI_L1C_TL_SGS__20151227T161254_
        A002679_T31TFK_N02.01/QI_DATA/S2A_OPER_MSK_NODATA_SGS_
        _20151227T161254_A002679_T31TFK_B01_MSIL1C.gml</path>
    <link>https://scihub.copernicus.eu/dhus/odata/v1/Products('
        b28dd648-55ef-45fd-951f-d8259b745b67')/Nodes('S2A_OPER_PR
        D_MSIL1C_PDMC_20151228T100614_R108_V20151227T104738
        _20151227T104738.SAFE')/Nodes('GRANULE')/Nodes('S2A_OPER_
        MSI_L1C_TL_SGS__20151227T161254_A002679_T31TFK_N02.01')/N
        odes('QI_DATA')/Nodes('S2A_OPER_MSK_NODATA_SGS__20151227
        T161254_A002679_T31TFK_B01_MSIL1C.gml')/$value</link>
    <checksum>310e42ac8018097e80054ae55900aae9</checksum>
    <status>checksum ok</status>
  </entry>
  <entry>
    <path>./GRANULE/S2A_OPER_MSI_L1C_TL_SGS__20151227T161254_
        A002679_T31TFK_N02.01/QI_DATA/S2A_OPER_MSK_DETFOO_SGS_
        _20151227T161254_A002679_T31TFK_B01_MSIL1C.gml</path>
    <link>https://scihub.copernicus.eu/dhus/odata/v1/Products('
        b28dd648-55ef-45fd-951f-d8259b745b67')/Nodes('S2A_OPER_PR
        D_MSIL1C_PDMC_20151228T100614_R108_V20151227T104738
        _20151227T104738.SAFE')/Nodes('GRANULE')/Nodes('S2A_OPER_
        MSI_L1C_TL_SGS__20151227T161254_A002679_T31TFK_N02.01')/N
        odes('QI_DATA')/Nodes('S2A_OPER_MSK_DETFOO_SGS__20151227
        T161254_A002679_T31TFK_B01_MSIL1C.gml')/$value</link>
    <checksum>a215f9fafed53c943f17cf3a9ae4e856</checksum>
    <status>checksum ok</status>
  </entry>
</feed>
```

FIGURE 6 – Rapport xml détaillé d'un produit dans la cas du téléchargement d'une archive par partie. Par souci de lisibilité, le rapport ne contient que 3 éléments.

spécifique à l'environnement linux), ce code ne fonctionne actuellement qu'avec Linux.

4.2 Librairies nécessaires

Deux librairies sont susceptible d'être manquante : lxml [6] et dateutil [7].

Le package lxml est utilisé afin de réaliser des entrées/sorties de fichier texte au format xml. La procédure d'installation est expliquée sur le lien suivant : http://lxml.de/installation.html.

Le package dateutil permet de traiter les différents format de date. La procédure d'installation est expliquée sur la réponse acceptée du lien stackoverflow suivant : http://stackoverflow.com/questions/20853474/importerror-no-module-named-dateutil-parser?lq=1. La réponse est copiée ci-dessous.

```
"On Ubuntu you may need to install the package manager pip first:
sudo apt-get install python-pip
Then install the python-dateutil package with:
sudo pip install python-dateutil"
```

Les autres librairies nécessaire devraient être installées par défaut.

4.3 Crontab

Comme évoquer précédemment, ce script est destiné à être exécuter régulièrement. La première possibilité est de lancer le script de manière manuelle en exécutant le script *main.py* dans le terminal ou dans n'importe quel IDE. Il est également possible de double cliquer sur le bash script *job_linux.sh* qui lui même exécute le script *main.py*.

Afin d'automatiser le lancement régulier du script, il est possible de programmer une tâche à l'aide d'une table de planification [8] sous linux. Dans ce cas il faut lancer le bash script *job_linux.sh* et non pas le *main.py*. En effet *job_linux.sh*, contient une protection permettant de lancer le *main.py* si et seulement si la dernière instance est terminé. Autrement dit, *job_linux.sh* ne permet de lancer qu'une seule instance du script à la fois. Afin de programmer une tâche, il suffit de lancer dans le terminal la commande *$ crontab -e* afin d'ouvrir la table et d'ajouter une tâche comme montré sur la figure 7 en remplaçant <*chemin*> par le chemin du dossier ou se trouve *job_linux.sh*. Dans cet exemple, le script est lancé tout les jours à 20h.

4.4 Exemple d'utilisation

Le fichier texte de requête (voir figure 8) contient deux exemples de requête sur la même zone d'étude (voir figure 9) et pour chaque satellite (Sentinel-1 et Sentinel-2). Ces requêtes sont ciblées sur le bassin de Barcelonnette dans les

6. http://lxml.de/
7. https://dateutil.readthedocs.org/en/latest/
8. https://doc.ubuntu-fr.org/cron

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirecte
    d).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cro
    n(8)
#
# m h  dom mon dow   command
#MAILTO="email@blabla.com"
0 20 * * * <chemin>/job_linux.sh
```

FIGURE 7 – exemple de programmation d'une tâche à l'aide de crontab de linux. Dans cet exemple, le script de téléchargement est lancé tout les 20h de chaque jour.

```
zone_etude;requete;nuage;Tiles;bands
Barcelonnette;platformname:Sentinel-2 AND footprint:"Intersects(P
    OLYGON((6.55 44.31, 6.55 44.47, 6.86 44.47, 6.86 44.31, 6.55
    44.31)))";25;T32TLQ;B02, B03, B04
Barcelonnette;platformname:Sentinel-1 AND footprint:"Intersects(P
    OLYGON((6.55 44.31, 6.55 44.47, 6.86 44.47, 6.86 44.31, 6.55
    44.31)))" AND beginposition:[2016-03-15T00:00:00.000Z TO NO
    W];;;
```

FIGURE 8 – Fichier requête exemple.

Alpes du Sud françaises grâce à la définition d'un rectangle défini en coordonnées (longitude, latitude) dans la requête OpenSearch. Pour Sentinel-2, un maximum de couverture nuageuse de 20% est défini et seul la tuile T32TLQ et les bandes B02, B03, et B04 sont récupéré. Pour Sentinel-1, tout les produits (tout niveau confondu) dont la date de début d'acquisition est comprise entre le 15 mars 2016 et la date d'aujourd'hui, sont récupérés.

Pour lancer le téléchargement de cet exemple, il faut suivre les étapes suivante :

1. Changer la valeur du mot de passe et de l'utilisateur du scihub dans le fichier de configuration config.cfg.
2. Changer la valeur du chemin de téléchargement dans le fichier de configuration config.cfg.
3. Exécuter *main.py* ou *job_ linux.sh* pour lancer le script de téléchargement.

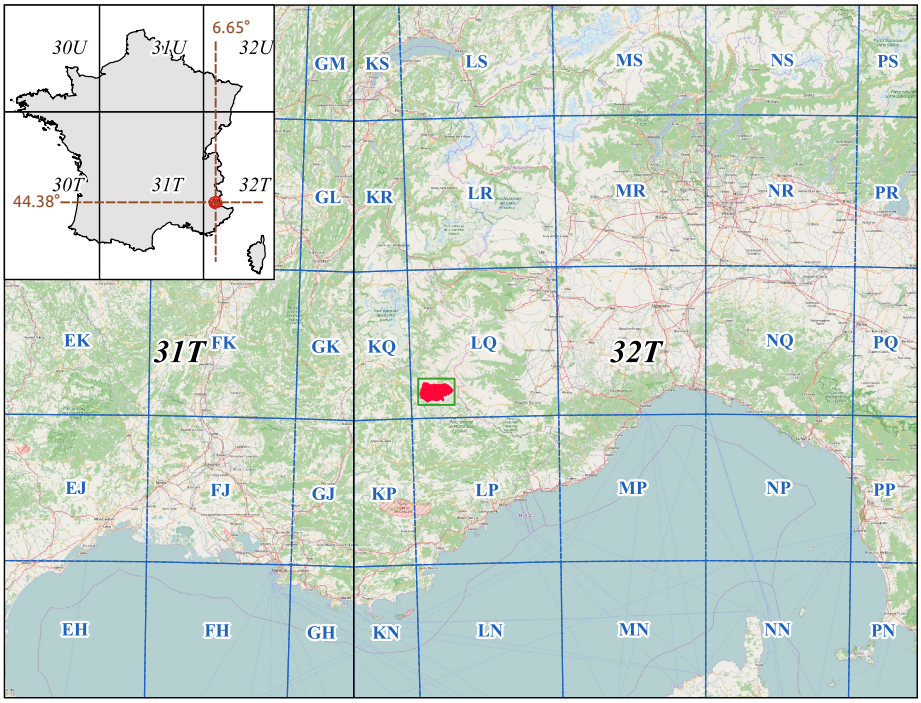

FIGURE 9 – Zone d'étude de Barcelonnette représenté en rouge. Le rectangle représenté en vert et qui entoure Barcelonnette a pour coordonnées (longitude, latitude), $\{(6.55, 44.31) ; (6.55, 44.47) ; (6.86, 44.47) ; (6.86, 44.31)\}$. Ce rectangle est utilisé dans la requête OpenSearch afin de ne récupérer que les produits de l'archive qui intersectent ce rectangle. On peut également voir que la zone d'étude de Barcelonnette n'est située que sur la zone 32TLQ du système MGRS.