DOCUMENT TECHNIQUE POUR LA MISE EN PLACE ET LA MAINTENANCE DE VERDANSONPLUIE

Observatoire Urbain HSM

Junior Muyumba Octobre 2023

Table des matières

1 Introduction

Le présent document aborde les aspects techniques relatives à la mise en place de VerdansonPluie, outil dédié à la visualisation des données des capteurs météo implantés dans le bassin versant de Verdanson par l'observatoire urbain.

Il reprend toutes les étapes d'installation et/ou configuration des programmes, logiciels nécessaires pour l'obtention de l'outil VerdansonPluie. Il énumère les différents codes écrits pour la collecte automatisée des données, les traitements ainsi que la diffusion de ces dernières, tout en fournissant plus ou moins les détails sur les fonctionnements de chaque script.

La chaine de traitement qui permet d'obtenir l'outil VerdansonPluie est mise en place sur base de deux projets créés sous deux langages à savoir python et R. Le projet ici renvoie un à répertoire contenant des scripts écris sous le langage concerné, ainsi que certains fichiers nécessaires aux fonctionnements des codes.

2 Présentation des projets

2.1 Le projet sous python

La figure 1 ci-dessous présente la structure du projet python nommé **observHSM** (Pour observatoire urbain HSM).

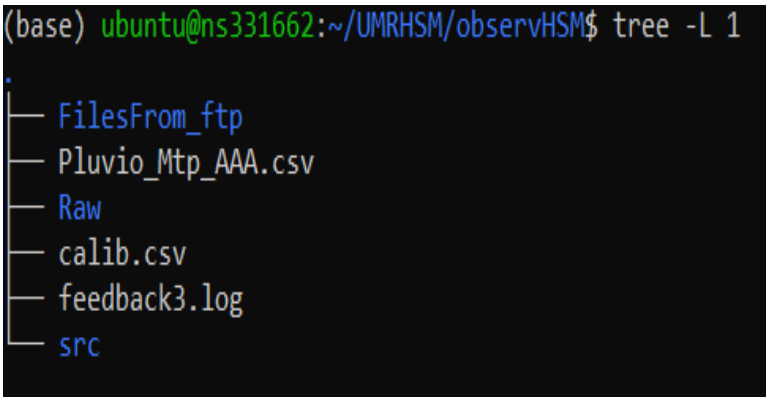

Figure 1 Projet observHSM

Ce projet contient des sous-répertoires (figure 2) contenant à leur tour, des fichiers des données brutes ou traitées, des scripts dont les détails sont donnés plus loin ainsi que d'autres fichiers connexes.

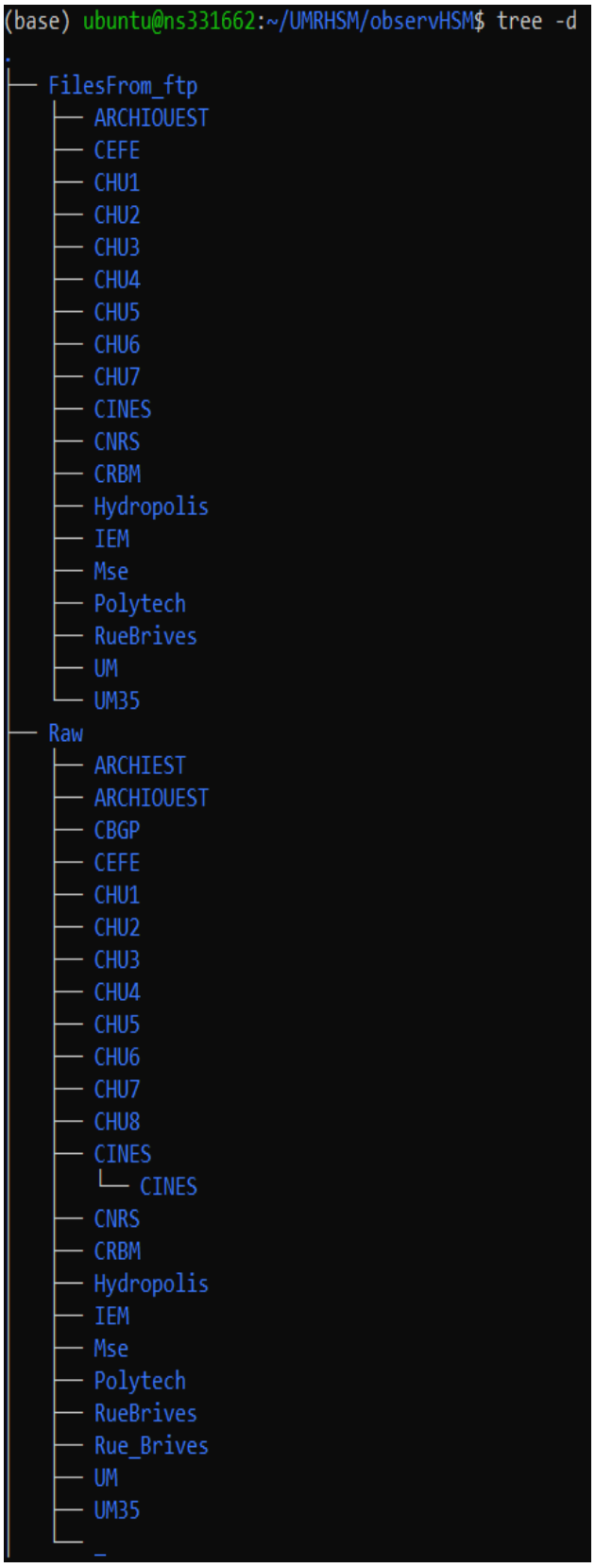

Figure 2 Sous répertoires du projet observSHM

- **Raw** : Répertoire contenant les fichiers des données brutes issues des capteurs qui arrivent chaque 5 min. Ces fichiers sont regroupés et stockés par station voire figure 3 ci- contre :

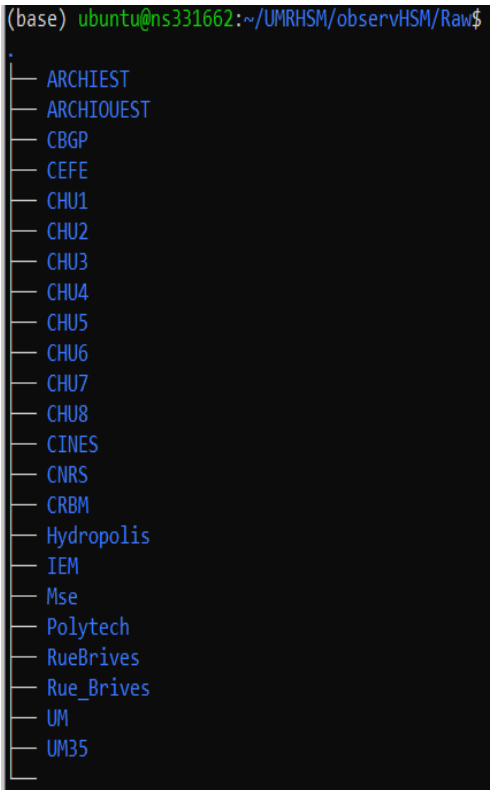

Figure 3 Raw - contient les données brutes

Le répertoire RAW est le double du répertoire ftp://ftp.hydrosciences.org/divers/Pluvio_Urbain/ il constitue donc la sauvegarde.

- **FilesFrom_ftp** (Nom ambigu peut-être qui pourra être modifié): a la même structure des fichiers, les mêmes noms et nombres des sous répertoires que **RAW** et **data** voir figure 3 ci-dessus. La différence réside sur les formats et niveau de traitement des données des fichiers qu'ils contiennent. FilesFrom_ftp contient les données déjà calibrées et non arrondies. C'est de ce répertoire que sont calculés les cumuls qui s'affichent dans l'appli VerdansonPLuie, et c'est de ce même répertoire que sont renvoyer les données vers le répertoire ftp://ftp.hydrosciences.org/divers/FromH2u/.

- **Calib**.csv : est simplement le fichier qui contient les coefficients de calibration des données.

NB : Ce fichier devra être actualiser à chaque ajout d'un nouveau capteur, c'est-à-dire y renseigner le nom, coordonnées géographiques et paramètres de calibration du nouveau capteur.

- **Feedback3**.log : la chaine de traitement s'exécutant en continue, un rapport d'erreur étant utile pour dénicher le problème en cas de disfonctionnement, ce fichier reçoit tous les messages d'erreurs qui surviennent lors de l'exécution.
- **Src** : est le répertoire qui contient tous les codes de collecte et traitement des données.

La figure 4 ci-dessous présente la liste des scripts se trouvant dans le répertoire src .

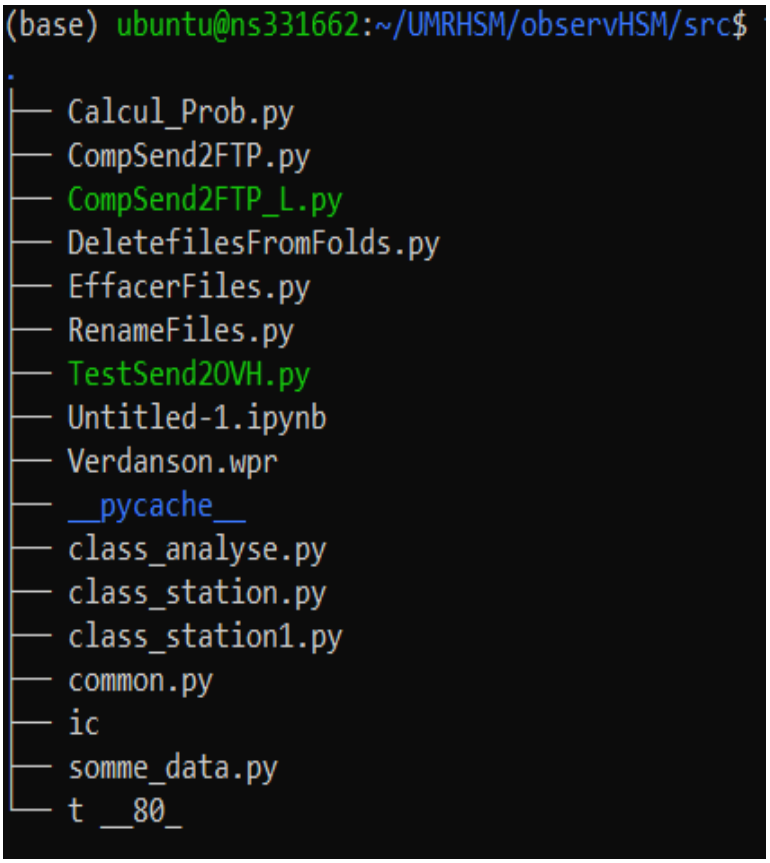

Figure 4 Src - Contient scripts

Les différents scripts seront détaillés plus loin en termes des variables et fonctions qu'ils contiennent.

2.2 Le projet sous R

La figure 5 ci-dessous présente la structure du projet R nommé **VerdansonPluie,** qui est l'application shiny même. Pareil que pour le projet observSHM, il contient des éléments nécessaires au bon fonctionnement de l'appli il s'agit ici du script R, des fichiers csv et un sous répertoire www.

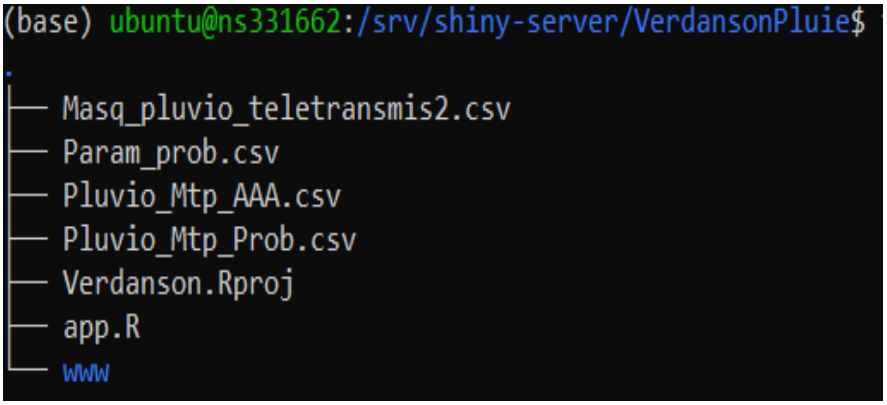

Figure 5 Projet VerdansonPluie

- **app.R :** C'est le script qui lance l'application VerdansonPluie, détails plus loin et aussi en commentaires au sein même du script.
- **Masq_pluvio_teletransmis2**.csv : est le fichier qui contient les coordonnées permettant de créer le masque ou la fenêtre d'observation.
- **Pluvio_Mtp_AAA**.csv : est le fichier qui contient les valeurs des cumuls.
- **Pluvio_Mtp_Prob**.csv : est le fichier contenant les estimations de la probabilité de non dépassement d'un cumul de pluie pendant la durée d (en minutes).
- **Param_prob** : est le fichier qui contient les coefficients pour les calculs des estimations de la probabilité de non dépassement d'un cumul de pluie pendant la durée d (en minutes).

NB : Les 5 paramètres fourni par Luc N. (a,b, μ, σ, ξ) variant suivant le pluvio, devront être tenu à jour, c'est-à-dire rajouter une fois obtenu. Car pour les capteurs qui en manquent actuellement, la fonction Cal_Prob renvoi simplement Null.

- **Verdanson**.Rproj : Ce fichier contient diverses options du projet et peut également être utilisé comme raccourci pour ouvrir le projet directement à partir du système de fichiers.
- **www** : est un sous répertoire contenant tous les fichiers d'images tels le logo, le label (exemple ici il contient le fichier png des étiquettes des pluviomètres utilisées dans l'appli).

NB : Contrairement au projet **observSHM** qui peut être créer n'importe où, le projet (répertoire) **VerdansonPluie** devra l'être dans le répertoire **/serv/shiny-server/** pour que l'appli soit accessible au serveur web apache2 et qu'elle soit mis à disposition du public.

3 Installations ou mises à jour et configurations à faire

Cette partie liste les installations et configurations des programmes et logiciels nécessaires à faire, afin de garantir le bon fonctionnement de l'outil VerdansonPluie.

3.1 Installation d'Apache

Apache : le serveur web apache2 est la première installation à faire, étant donné que l'application VerdansonPluie est destinée à être mis en ligne, sans quoi cela ne sera pas effectif. Pour installer le serveur apache, il suffit d'exécuter dans le terminal les deux commandes ci-dessous :

```
sudo apt update
sudo apt install apache2
```

Une fois l'exécution terminée, vérifier en tapant « localhost » dans n'importe quel navigateur présent sur l'ordinateur. Si l'installation s'est faite avec succès, une page telle que présentée à la figure 6 ci-dessous devra s'afficher.

Figure 6 Apache2 Works

3.2 Installations pour le projet python

Python : La quasi-totalité des prétraitement et traitement des données se fait quasiment à l'aide du langage python. Cependant, aucune installation du langage n'est nécessaire étant donné que la distribution Linux Ubuntu s'installe par défaut avec une version de python. A la date d'aujourd'hui, la version Ubuntu 23.04 propose par défaut la version 3.11.4 de python. La seule raison de pouvoir installer une nouvelle version serait le besoin d'avoir une version spécifique. A noter que l'obtention d'une version spécifique de python, est facilitée par le gestionnaire anaconda, qui offre la possibilité de créer plusieurs environnements python avec diverses versions du langage.

Pycharm : Est un IDE qui offre la saisie de code intelligente, des inspections de code, la mise en évidence des erreurs à la volée et des correctifs rapides, en plus des refactorisations de code automatisé et de riches capacités de navigation. Son utilisation requière avoir java et un interpréteur Python (2 ou 3) installés. L'installation de pycharm-community (la version gratuite et suffisante pour les tâches concernant le projet), pourra se faire soit via l'interface directement dans le logiciel « Ubuntu software » ou via la ligne de commande tel qu'indiqué ici. Ou tout simplement avec snap, exécuter dans le terminal la commande ci-dessous :

```
sudo snap install pycharm-community --classic
```

Anaconda : ou conda, est un système de gestion des paquets et de l'environnement open-source qui fonctionne sous Windows, macOS et Linux. Conda installe, exécute et met à jour rapidement les paquets et leurs dépendances. Il crée, enregistre, charge et passe facilement d'un environnement à l'autre sur votre ordinateur local. Il a été créé pour les programmes Python, mais il peut empaqueter et distribuer des logiciels pour n'importe quel langage (ex pycharm cité ci-haut , rstudio..). Son installation peut se faire aisément comme cela est montré ici soit ici ou encore en exécutant les commandes ci-après :

- `sudo apt update`

- `sudo apt install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1 libxcomposite1 libasound2 libxi6 libxtst6`

- `wget -P /tmp https://repo.anaconda.com/archive/Anaconda3-2020.02-Linux-x86_64.sh`

Afficher la somme de contrôle du script avec la commande ;

- `sha256sum /tmp/Anaconda3-2020.02-Linux-x86_64.sh`

Le résultat devrait ressembler à ceci :

- `2b9f088b2022edb474915d9f69a803d6449d5fdb4c303041f60ac4aefcc208bb /tmp/Anaconda3-2020.02-Linux-x86_64.sh`

Exécutez le script pour lancer le processus d'installation :

- `bash /tmp/Anaconda3-2020.02-Linux-x86_64.sh`

Appuyer sur ENTER pour continuer, approuver les termes de la licence en tapant yes et accepter l'emplacement de l'installation exemple :

/home/hsm/anaconda3

Appuyer sur ENTER pour confirmer l'emplacement ;

L'installation terminée, autoriser au programme d'installation d'initialiser Anaconda3. Ensuite charger la nouvelle variable d'environnement PATH dans la session shell en cours en tapant :

- `source ~/.bashrc`

Enfin, mettre à jour anaconda en entrant dans le terminal

- `conda update --all`

Anaconda-navigator : Anaconda Navigator est une interface graphique de bureau (GUI) incluse dans Anaconda® Distribution qui permet de lancer des applications et de gérer les paquets conda sur Anaconda.org ou dans un dépôt local d'Anaconda, les environnements… Il est disponible pour Windows, macOS et Linux. Peut s'installer avec la commande `conda install anaconda-navigator` pour en savour plus sur la configuration, lire.

Pip : Pareil que Conda, PIP est un gestionnaire de paquets ou modules Python. Note : A partir de la version 3.4 de Python à la plus récente, PIP est inclus par défaut. Il peut être mis à niveau en exécutant dans le terminal la commande :

- `python3 -m pip install –upgrade pip`

Avec tous les programmes et logiciels ci-dessus, nous sommes en mesure de faire marcher correctement tous les codes python du projet.

3.3 Installations pour le projet R

R : L'installation de R peut se faire en ligne de commande en exécutant successivement les commandes qui suivent.

- `sudo apt install dirmngr gnupg apt-transport-https ca-certificates software-properties-common`

- `sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9`
- `sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu focal-cran40/'`
- `sudo apt install r-base` ou `sudo apt-get install r-base-dev=3.6.3-2` si l'on souhaite installer une version spécifique de R.
- `R –versionn`
- `sudo apt install build-essential`

Enfin la commande

- `R`

S'il s'affiche un message indiquant la version de R et un prompte qui attend de recevoir les commandes, ce que l'installation s'est faite avec succès. A ce stade il ne restera qu'à installer les packages utiles pour travailler.

L'installation d'un package peut se faire à partir de Rstudio, ou en ligne de commande en exécutant la commande « `install.packages("Nom_du_package")` », après avoir déjà lancer R bien entendu. Afin de palier au problème de non accessibilité des packages par certains utilisateurs, il est recommandable de lancer au préalable R en mode super user c'est-à-dire
« `Sudo R` » Sinon, il y a risque d'obtenir l'erreur du type « *The following object is masked from 'package: Nom_du_package': »*

NB : une autre manière d'installer les packages, très recommandable est d'utiliser la commande ci-après.

- `sudo su - -c "R -e \"install.packages('Nom_du_package',repos='http://cran.rstudio.com/')\""`

Exemple :

- `sudo su - -c "R -e \"install.packages('leaflet', repos='http://cran.rstudio.com/')\""`

RStudio devenu maintenant **Posit PBC :** Posit(Rstudio) pour à R ce que pycharm est pour python. RStudio intègre la possibilité d'écrire des notebooks combinant de manière interactive du code R, du texte mis en forme en markdown et des appels des codes Python ou Bash[5]. Pour l'installer voir ici.

3.4 Les configurations

Afin de rendre l'application VerdansonPluie accessible à tout public de le faire fonctionner continuellement, quelques configurations présentées ci-dessous s'imposent.

3.4.1 Configuration Shiny-server

Le client de l'application VerdansonPluie s'obtient grâce à shiny-server. Ce dernier nécessite quelques configurations afin de garantir le bon fonctionnement. Pour le faire, lancer successivement les commandes ci-dessous :

- `sudo su - -c "R -e \"install.packages('shiny', repos='http://cran.rstudio.com/')\""`

- `wget https://download3.rstudio.org/ubuntu-18.04/x86_64/shiny-server-1.5.18.987-amd64.deb`

- `sha256sum shiny-server-1.5.18.987-amd64.deb`

Ces trois premières commandes à lancer si shiny n'est pas encore installé.

- `sudo apt update`

- `sudo apt install gdebi-core`

- `sudo gdebi shiny-server-1.5.17.973-amd64.deb`

- `sudo ss -plut | grep -i shiny`

- `sudo ufw allow 3838` ; cette commande permet à Firewall d'autoriser le port 3838 à partir duquel l'application sera accessible en ligne.

- `sudo su - -c "R -e \"install.packages('rmarkdown', repos='http://cran.rstudio.com/')\""`

Par défaut, il est configuré pour servir les applications dans le répertoire /srv/shiny-server/, ce qui signifie que toute application Shiny placée dans /srv/shiny-server/app_name sera accessible au public à l'adresse https://example.com/app_name/ ou http://Adress_IP:3838/sample-apps/rmd/.

Si l'installation s'est bien faite, un exemple d'application shiny est accessible au public. Exemple à l'adresse : http://hydro-h2u.msem.univ-montp2.fr:3838/sample-apps/rmd/

En cas de non accès à l'application shiny, il sera nécessaire d'apporter une modification au fichier de configuration « shiny-server.conf » situé dans le répertoire /etc/shiny-server/. Elle consistera à remplacer dans le fichier "run_as shiny » voir figure 7 par « run_as user" (Pour notre exemple ce sera "run_as ubuntu »).

```
# Instruct Shiny Server to run applications as the user "shiny"
run_as ubuntu;
app_init_timeout 300;

app_idle_timeout 200;
# Define a server that listens on port 3838
server {
  listen 3838;

  # Define a location at the base URL
  location / {

    # Host the directory of Shiny Apps stored in this directory
    site_dir /srv/shiny-server;

    # Log all Shiny output to files in this directory
    log_dir /var/log/shiny-server;

    # When a user visits the base URL rather than a particular application,
    # an index of the applications available in this directory will be shown.
    directory_index on;
  }
}
```

Figure 7 shiny-server.conf

Le fichier de configuration n'étant pas directement éditable, il est primordial de changer le droit d'accès en procédant comme suit :

Pour accéder au répertoire contenu le fichier

- `cd /etc/shiny-server/` =>.conf

Changer le droit d'accès et d'écritures.

- `sudo chown hsm.hsm shiny-server.conf`

- `sudo vi /etc/shiny-server/shiny-server.conf` = > Ouvrir le fichier pour commencer l'édition du fichier. Enregistrer les modifications faites (dans vi avec la commande : wq).

Ensuite, dire à firewall d'autoriser le port 3838 :

- `sudo ufw allow 3838/tcp`

!!! Attention : n'est pas oublié de remettre le droit d'accès et d'écriture à root :

- `sudo chown root.root shiny-server.conf`

Enfin, Une fois la modification faite, redémarrer shiny_server en tapant :

- `sudo systemctl restart shiny-server.service`

Avec tout ce qui précède l'application shiny déployée fonctionnera normalement.

NB : Si tout de même il s'affiche le message ci-dessous dans le navigateur :

« An error has occurred

The application failed to start.

The application exited during initialization. »,

Cela n'est pas si grave, ce que shiny ne parvient pas à se relancer par exemple suite à un problème d'insuffisance d'espace disque. Pour remédier dans ce cas, il suffit de disponibiliser l'espace sur la machine ensuite relancer shiny avec la commande : `sudo systemctl restart shiny-server.service`

3.4.2 Configuration de Cron

L'application étant destinée à fonctionner en continue, il est impérieux de configurer l'automatisation d'exécutions des tâches. Sur l'OS Linux, Cron est le programme qui permet aux utilisateurs de programmer l'exécution d'un logiciel, il est généralement utilisé lorsqu'une tâche doit être exécutée selon un calendrier fixe, et/ou pour automatiser des tâches répétitives. Pour en savoir plus sur comment lancer les tâches ou encore comment personnaliser le lancement d'une tâche avec le programme cron lire ici. Il suffit de taper dans le terminal la commande crontab -e.

NB : Dans Cron, une commande précédée d'un # est un commentaire,

elle ne sera donc pas exécutée.

Exemple d'un cron job :

```
#*/5****/usr/bin/python3/home/hsm/Bureau/UMRHSM/obserHSM/src/
CompSend2FTP_L.py >/home/hsm/Bureau/UMRHSM/obserHSM/feedback4.log
2>&1
```

Cette commande ci-dessus veut juste dire que le code CompSend2FTP_L.py est mis en commentaire ici. Il s'exécuterait toutes les 5 minutes s'il ne l'était pas. Et en cas des difficultés d'exécution, les messages d'erreur seraient redirigés vers le fichier feedback4.log, ce qui est pratique pour déboguer.

Pour ce qui concerne la chaine de traitement de VerdansonPluie la commande cron qui est utilisée pour faire tourner chaque minutes le code est :

```
*/1 * * * * /usr/bin/bash /home/hsm/scriptWait3.sh > /home/hsm/feedback3.log 2>&1 1
```

Cette commande s'exécute chaque minute, mais ne relance le code CompSend2FTP_L.py que sous condition que le job précédent soit terminé ou encore qu'il n'y ait qu'un nombre inférieur ou égale à 4, des codes python qui tournent simultanément (sinon il attend). Ce qui est pratique pour éviter la saturation du processeur. Elle renvoie les messages d'erreurs dans le fichier feedback3.log (figure 1)

NB : Pour lancer un code dans cron, il est indispensable de le rendre préalablement exécutable et cela peut se faire en :

1- Ajoutant un shebang dans le script à la première ligne.
   - `#! interpreter [optional-arg]`

`Exemples :`

`Shebang dans un script python exécutable =➔ #!/usr/bin/env python3`
`Shebang dans un script shell exécutable =➔ #!/bin/bash ;`

2- Rendant exécutable le script par la commande : `chmod +x Nom_script`

Example :

   - `chmod +x CompSend2FTP_L.py`

Puis vérifier si la procédure s'est faite avec succès en tapant :

   - `./CompSend2FTP_L.py`

Remarque Cron :
Pour que les données soient à jour et que VerdansonPluie affiche des valeurs des cumuls corrects, il est impératif que cron exécute continuellement les commandes qui lui sont données. Dans le cas contraire aucune nouvelle donnée ne s'affichera sauf exécution manuelle du script `scriptWait3.sh` se trouvant à la racine comme ceci : (base) ubuntu@ns331662:~$ `./ scriptWait3.sh`

Il faudra retenir qu'ayant confié à cron la tâche d'exécuter chaque fois nos codes, son disfonctionnement occasionnera certaines erreurs. Afin de les identifier et les résoudre, voir plus loin les consignes présentées aux points **4.1** et **4.2** de la partie 5 du présent document. Détails des scripts

3.5 Scripts python

Liste des scripts et fonctions qu'ils contiennent

- Calcul_Prob.py : code qui estime la probabilité de non dépassement d'un cumul de pluie pendant une durée donnée en minute.

- class_station.py : fait la calibration des données, convertit les pluies en mm et les vitesses de vents en km/h et crée le fichier des données calibrées.

- common.py : Lit les données en entrée et gère le problème des données manquantes.

- CompSend2FTP.py : Contient ses fonctions locales et celles importées des autres codes. C'est finalement ce script qui contient la majeure partie des fonctions et variables nécessaires pour le pré-traitement et traitement des données.

- somme_data.py : est la fonction qui calcul le cumul des pluies par station à 15, 30, 1h, 2h, 4h,12h,24h et 72h.

- CompSend2FTP_L.py (_L pour Launcher) : Afin d'éviter les chevauchements dans les exécutions des différentes tâches, ce script permet de lancer dans un ordre chronologique les tâches.

- DeletefilesFromfolds.py
- EffacerFiles.py
- RenameFiles

Ces trois précédents scripts sont **d'usages ponctuelles** mais pratiques pour supprimer ou renommer par exemple tous les fichiers de l'ensemble des sous-répertoires.

Ici nous détaillons les deux fonctions et variables contenus dans les scripts Calcul_Prob.py et CompSend2FTP.py car, les détails des autres peuvent -être retrouver en commentaire au sein des scripts.

3.5.1 CompSend2FTP.py

#!/usr/bin/env python3 / ➔ Sheebang à la ligne initiale du code.

3.5.1.1 Librairies

- Ftplib, os,sys,os.path,glob,datetime,class_station,common,csv,math…

3.5.1.2 Variables et roles

- Start : récupère l'heure du début d'exécution du programme
- Data_path : c'est la variable qui définit le répertoire ou sont situés les fichiers CSV calibrés de l'année en et qui sont complété progressivement, ces fichiers prêts à être renvoyer sur le FTP. Cette variable est donc utilisée dans la fonction send2Ftp, elle est aussi utilisée dans la fonction delete_f

- Tz : cette variable permettait d'obtenir le temps en heure locale. Cependant, depuis la décision de laisser les données en TU, il ne sert plus et voilà pourquoi elle est mise en commentaire.

- Station_list : cette variable contient la liste de tous les capteurs. A partir d'elle sont

effectuées bon nombre des tâches. Il est impérieux de placer à la même indice chaque nom du capteur présent dans cette liste et ses coordonnées géographiques x et y correspondant dans la liste Coord_geo. Car ces trois éléments (nom du capteur et ses deux coordonnées sont récupérés conjointement et utilisés par exemple dans somme_data.py et Calcul_Prob.py pour respectivement calculer les cumuls et les estimations de la probabilité de non dépassement d'un cumul de pluie.

Elle est utilisée dans les fonctions ci- après :

- fromFtp : à partir de la liste, la fonction permet de parcourir les répertoires se trouvant sur le Ftp et récupère ces fichiers et le stock en local dans les répertoires propres à chaque capteur. Avant de télécharger les fichiers, la fonction vérifie qu'il existe bel et bien un répertoire pour chaque capteur, sinon elle en crée.
- Calibrating2 : la fonction crée des fichiers des données calibrées dans les répertoires respectifs si ces derniers existent, sinon elle en crée. NB : il s'agit de deux chemins différents qui sont créées les répertoires par les deux fonctions. fromFtp crée le répertoire d'entrée, alors que calibrations2 crée les répertoires des fichiers sur des données calibrées, sur lesquels porte la suite des manipulations, ce sont ces répertoires qui contiennent les fichiers qui seront renvoyer sur le serveur Ftp dans le dossier FromH2U.

- Send2Ftp : cette fonction renvoie les fichiers calibrés vers le répertoire FromH2U du Ftp.
- delete_f : pour une raison ou une autre, il peut y avoir besoin de supprimer les fichiers contenus dans les divers sous-répertoires(qui correspondent aux dossiers capteurs contenant les données calibrées) dans le répertoire **FilesFrom_ftp**, sans avoir à parcourir chaque répertoire. Cette fonction automatise la tâche. Elle nécessitera bien entendu des adaptations.

- Coord_geo : cette variable est une liste qui contient les coordonnées géographiques des stations, elle a deux sous listes respectivement les longitudes et les latitudes. Elle est utilisée dans le dictionnaire dict_dat, ce dernier est un modèle dans lequel seront remplis les cumuls des pluies et à partir duquel sera ainsi crée le fichier des cumuls lu par l'application VerdansonPluie.
- dir_fold_out : est la variable qui contient le chemin du répertoire où se trouve l'application VerdansonPluie et où sera écrit le fichier de cumules des pluies.
- path_in : répertoire parent où sont créés par la fonction calibrating2, les répertoires des fichiers des données calibrées.
- Name : est simplement le préfixe du nom du fichier des cumules à créer.
- Dict_dat : déjà expliqué ci-haut.
- annee_en_cours : comme son nom l'indique, cette variable contient l'année en cours. Elle est utilisée dans la fonction ReadRawData qui lit les données par rapport à la date, ainsi que dans la fonction calibrating2.

- Mois_en_cours : comme pour la variable précédente, elle contient le mois en cours et se trouve dans la fonction RedRawData.
- aujourd8 : contient le jour d'aujourd'hui et se trouve dans la même fonction que la variable précédente. Cette variable permet par exemple est utilisée pour récupérer les données à une durée voulue, exemple les données télétransmis il y a deux jours c'est-à-dire données à (aujourd8 - 2)
- end : récupère l'heure de la fin d'exécution du programme
- deltatT : contient la différence du temps de la fin de celui du début d'exécution du programme
- annee
- NB : la variable annee est différente de la variable annee_en_cours. Par exemple pour l'année 2023, annee = _2023 et annee_en_cours = 23

3.5.1.3 Les fonctions

- fromFtp : écrit en se basant sur les librairies ftplib, os et os.path. Elle établit la connexion avec le serveur FTP ici matérialisé par l'objet ftp_hsm, crée un dossier par capteur et y écris tous les nouveaux fichiers correspondants téléchargé sur le ftp.
- Class_station : est la librairie qui contient la fonction Calibrating2 par exemple

- Calibrating2 : Cfr explication ci-haut
- readCalib : La fonction readCalib issus de class_station et basé sur la librairie csv , lu le fichier csv de calibration et retourne les paramètres nécessaires pour la calibration.
- ReadRawData : fonction de lecture des données brutes, il peut lire que les données d'il y a 1, 2, …x nombre des jours selon la configuration souhaitée.

3.5.2 Calcul_Prob.py

3.5.2.1 Librairies

- datetime, pandas, math, numpy.

3.5.2.2 Variables

- Station_list et coordo_geo, explication cfr. CompSend2FTP.py ci-haut.
- dir_fold_out : le chemin où sera écrit le fichier csv contenant les estimations de la probabilité de non dépassement pendant la durée d (en occurrence le fichier Pluvio_Mtp_Prob.csv voir figure 5).
- path_to_cumulfiles : chemin du fichier csv contenant les cumuls des pluies.
- name : Juste une variable qui contient le nom à donner au fichier contenant estimations de la probabilité de non dépassement pendant la durée d (en minutes).
- Path_to_param_file : chemin du fichier en entrée contenant les coefficients utilisés dans les calculs des estimations de la probabilité de non dépassement d'un cumul de pluie pendant la durée **d** (en minutes).

3.5.2.3 Fonctions

- Cal_prob : cette fonction contient la formule fournit par Luc N. pour les calculs des estimations de la probabilité de non dépassement d'un cumul de pluie pendant la durée d, à partir des cumuls et des coefficients fournit.

NB : Ces calculs ne sont faits que sur des durées comprises entre 1 min < d < 240 min (4h) et que pour de la pluie c'est -à-dire en cas de valeur nulle ou absence de valeur la fonction retourne la valeur **'nan'.**

- Cal_PR_Prob : est la fonction qui permet d'appliquer la fonction précédente à toutes les valeurs des cumuls et produit en sortie le fichier « Pluvio_Mtp_Prob.csv » contenant les estimations de la probabilité de non dépassement d'un cumul de pluie pendant la durée d (en minutes).

3.6 Script R

3.6.1 Les librairies R

- Shiny, leaflet, shinyscreenshot, gstat, raster, FRK, rgdal, png

Les descriptions existante commentaire dans le code.

4 Prévention ou résolution du bug à la relance des scripts après un long arrêt

Au cas où Cron n'aurait pas exécuté le script `scriptWait3.sh` et que par conséquent, le téléchargement des fichiers, les calculs des cumuls et des estimations de la probabilité de non dépassement ne seraient pas effectués. Il est recommandé à la prochaine relance, de supprimer tous les fichiers des données calibrées se trouvant dans tous les sous répertoires du répertoire **FilesFrom_ftp.** Pour se faire, il suffira de mettre en commentaire (en faisant précéder par un #) toutes les commandes du script **CompSend2FTP_L.py** en ne laissant que la dernière comme c'est-à-dire la ligne où est fait appel la fonction **delete_f**. En suite s'assurer que les configurations proposées au point **4.1** ci-dessous sont bien prises en compte avant une nouvelle exécution du script '`scriptWait3.sh`', ainsi seront réécrit les fichiers calibrés de l'année à l'instant (t) d'exécution.

L'inconvénient de garder indéfiniment la configuration proposée au point **4.1**, ce que la réécriture complète des fichiers à chaque exécution mettra chaque fois plus de temps. Ainsi, afin d'éviter que cette opération soit chronophage, après une première exécution suivant ce qui est proposé au point **4.1** il sera préférable d'apporter les modifications fournies au point **4.2** afin de n'ajouter à chaque fichier existant que les dernières données (à 1, 2,3 … jours de l'instant t ou s'exécute le programme).

!! Aucun souci à se faire concernant la répétitivité des manipulations, elles ne seront pas à refaire régulièrement. Normalement si Cron est bien configuré et si le droit d'administrateur son bien définit permettant à ce dernier de fonctionner sans contrainte et qu'il n'y ait pas disfonctionnement, les manipulations en **4.1** et **4.2** ne seront réalisées qu'une seule fois.

Déjà pour s'assurer que Cron fonctionne comme il faut, il peut être intéressant de faire par exemple une vérification du dernier instant auquel un nouveau fichier en provenance du capteur a été

ajouter dans le répertoire du capteur avec la commande ls -latr | tail -1.

Exemple : (base) ubuntu@ns331662:~/UMRHSM/observHSM/Raw/Mse$ ls -latr | tail -1
On a la sortie suivant : -rw-rw-r-- 1 ubuntu ubuntu 62 Jan 15 09:19 MSE_140124_020200.txt , ce qui montre que le tout dernier fichier provenant du capteur Mse fut ajouté au répertoire au portant le même nom en date du 15 janvier de l'année 2024 en cours, à 09h19 alors que cette vérification s'est faite le 19/01/2024. Ce qui atteste le non téléchargement des nouvelles données depuis 4 jours.

Afin d'effectuer les opérations recommander au point **4.1** ou **4.2** ci-dessous, il est impératif de se situer dans le répertoire ~/UMRHSM/observHSM/src/

4.1 Réécriture intégrale des fichiers de données

- Ouvrir le script class_station.py, dans la fonction write, s'assurer que le mode d'ouverture des fichiers est bel et bien **'w'**(pour write), dans l'instruction with pour ouvrir le fichier et écrire dedans. S'assurer aussi que dans la même fonction, la ligne writer.writerow(['Time_%Y-%m-%dT%H:%M:%S','Rainfall','Wind_velocity','Battery']) n'est pas mise en commentaire, si tel est le cas enlever le # devant.
- Dans le script CompSend2FTP, au sein de la fonction ReadRawData , vu que le souhait ici est de réécrire le fichier des données traitées dès le début de l'année, il faudra après la condition **if**, passer directement au **else**, c'est-à-dire garder en commentaire la condition **elif**.

4.2 Ajout des nouvelles données aux fichiers existants

- Ouvrir le script class_station.py, dans la fonction write, s'assurer de bien remplacer 'w' par **'a'** (pour append), dans l'instruction with pour ouvrir le fichier et ajouter à la suite. S'assurer aussi que dans la même fonction, la ligne writer.writerow(['Time_%Y-%m-%dT%H:%M:%S','Rainfall','Wind_velocity','Battery']) cette fois est commentée(#), pour pas avoir à rajouter les entêtes à chaque ajout des nouvelles lignes des données.
- Dans le script CompSend2FTP, au sein de la fonction ReadRawData : vu que le souhait est de juste rajouter les toutes dernières données aux fichiers existants, il faudra après la condition **if**, enlever # au début de la ligne de la deuxième condition **elif**. elif (filepath.split("_")[1][-2:]) == annee_en_cours and jr_fichier >= str(int(aujourd8)-2) and mois == mois_en_cours et par contre mettre **else** en commentaire. Sachant qu'ici suivant la condition elif, nous ne récuperons que les données d'il y a deux et les ajoutons au fichier. Sinon si l'on souhaiterait par exemple récupérer celles d'il y a 4 jours, il suffira de modifier la partie jr_fichier >= str(int(aujourd8)-2) en jr_fichier >= str(int(aujourd8)-4).

5 Remarque

Il est important de redire un mot sur les différences entre les scripts CompSend2FTP.py, CompSend2FTP_L.py et `scriptWait3.sh` **:**

Le premier est celui qui télécharge, traite les données et crée les fichiers des séries chronologiques

des données. Le deuxième et celui qui orchestre l'exécution chronologique des commandes du scripts **CompSend2FTP.py** et celles d'autres scripts tels **somme_data.py** (qui calcul les cumuls des pluies), et **Calcul_Prob.py** (qui calcul les estimations …). Alors que le troisième soit **scriptWait3.sh** permet de vérifier avant d'exécuter CompSend2FTP.py s'il n'y a pas plus de 4 programmes python qui s'exécutent simultanément au point de créer des problèmes de surconsommation des ressources. Ce qu'il faudra noter ce que ce script .bash était nécessaire par exemple lors qu'il y avait en parallèle d'autres traitement python pour istSOS. Mais sur le serveur OVH il n'y aura pas d'incidence de mettre directement **'CompSend2FTP_L.py'** dans un Cron-job, la raison d'utiliser scriptWait3.sh est juste préventive.

!!!! La question de comment savoir si un capteur n'envoie pas les données. Après test et observation, il était ressorti que le capteur ayant rencontré le problème de télétransmission des données n'apparaitra pas sur la page de VerdansonPluie. Cependant, si un capteur fonctionne mais n'a pas juste reçu de pluie, il affichera (0.00 ou NA).