Mémoire technique

Projet Scripting system

Sommaire

1 - Présentation générale du projet :

Il s'agit d'un projet réalisé en Python qui va être initialisé grâce au fichier config.ini regroupant différents paramètres qui devront obligatoirement être complétés au gré de l'utilisateur.

Il existe 2 scripts dans notre projet : le 1er (script.py) va servir de script principal : il va réaliser toutes les opérations en lien avec le fichier que l'on souhaite traiter. Le 2ème (inform.py) sert à informer des résultats qu'a obtenu le script via une notification mattermost ainsi qu'un courriel.

Le déroulement du projet consiste à télécharger un fichier zip depuis un serveur web.
Dézipper ce fichier afin d'obtenir un dump SQL (test_export.sql).
On va comparer ce dump à un autre de référence afin de déterminer s'il y a eu des modifications dessus depuis.
Une fois ces vérifications passées on va le compresser en format tar.gz puis l'envoyer à un serveur SMB (Windows) afin de l'archiver au format AnnéeJourMois (AAAADDMM.tar.gz).
Les progrès seront transmis sur mattermost et par courriel en utilisant le protocole SMTP.

On effectuera une exécution automatique du script de manière régulière en utilisant le planificateur de tâches de Windows.

L'arborescence est constituée de deux scripts python, un fichier log, un fichier txt servant à installer les dépendances nécessaires au bon fonctionnement du code, un fichier config.ini permettant à l'utilisateur de modifier les variables essentielles du code sans passer par le fichier python. Un dossier Reçu qui contient le dump SQL (test_export.sql) de référence.

2 - Structure

La structure du code est écrite sous forme de classes avec des attributs et des méthodes.
Cela nous permet non seulement d'avoir un programme avec une structure solide et clair mais aussi de pouvoir la modifier / ajouter des fonctionnalités assez simplement. Il serait même possible de pouvoir gérer plusieurs fichiers en instanciant plusieurs fois cette classe.

La classe script :

En attributs :

- Un fichier log (recap.log) : logger
- Un fichier de config (config.ini) : config

Tous les attributs suivants sont quant à eux responsable de déterminer le statut des différentes étapes du script :

- charg : indique si le fichier de config a bien été chargé
- validRec : indique si le fichier zip a bien été téléchargé
- validDump : indique si la présence du dump SQL dans le zip est vérifiée
- validPasPareil : indique si les fichiers ne sont pas similaires
- validCompression : indique si la compression s'est bien effectuée
- validConn : indique si la connexion s'est effectuée avec le serveur SMB
- validUpload : indique si le transfert de l'archive a bien été effectué

En méthodes :

- La méthode __init__ propre à python permet d'initialiser les attributs, correspond au constructeur
- zip_recup : télécharge le fichier zip à l'url indiqué dans le fichier config
- extraction_comparaison : Vérifie la présence du dump SQL dans le fichier zip, extrait et compare le dump avec celui de référence (si ce n'est pas la 1ère fois que le script est lancé)
- compression : compresse le dump SQL sous tar.gz au format AAAA/DD/MM
- envoi_smb : se connecte au serveur SMB distant, supprime les fichiers trop anciens (le nombre de jours max étant défini dans le fichier config) si l'historisation est activée, et envoi la nouvelle archive.

La classe envoi :

En attributs :

- Un fichier log (recap.log) : logger
- Un fichier de config (config.ini) : config
  Tous les attributs suivants sont quant à eux responsable de déterminer le statut des différentes étapes du script :
- charg : indique si le fichier de config a bien été chargé
- validRec : indique si le fichier zip a bien été téléchargé
- validDump : indique si la présence du dump SQL dans le zip est vérifiée
- validPasPareil : indique si les fichiers ne sont pas similaires
- validCompression : indique si la compression s'est bien effectuée
- validConn : indique si la connexion s'est effectuée avec le serveur SMB
- validUpload : indique si le transfert de l'archive a bien été effectué

En méthodes :

- La méthode __init__ propre à python permet d'initialiser les attributs, correspond au constructeur
- notif_mattermost : Créer un message en json incluant toutes les variables de statuts du script puis l'envoi au serveur mattermost grâce au webhook dans le fichier config.ini
- envoi_email : Génère un message en html, puis l'envoi par email avec comme pièce jointe le fichier log

3 - Description des méthodes :

La classe script :

__init__ :

Cette méthode prend en paramètre l'instance actuel self, le fichier log dans lequel on va écrire les différentes étapes qu'a franchi le script, le fichier de configuration.
On configure le module logging afin de pouvoir suivre les étapes de notre code (grâce au logger) sur notre invite de commande / terminal. On fait appel à un fileHandler pour pouvoir écrire ces résultats dans notre fichier log avec un certain format défini grâce au formatter.

On initialise tous les variables nous servant à indiquer les étapes pour la notification mattermost et email :
« :no entry : » correspondant à un symbole stop sur mattermost (indiquant qu'il y a un problème), plus on avancera dans le code plus on passera ces variables à la valeur « :white_check_mark: » symbolisant une étape réussie.

On passe désormais à la lecture du fichier de config grâce au module configparser. On vérifie que toutes les sections nécessaires ainsi que leurs options sont bien présentes. On indique chaque étape à chaque fois au logger.
Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur charg.

zip_recup :

On informe qu'on commence cette étape avec le logger. On vérifie la présence du dossier Recu sinon on le créer avec le module os.

En utilisant le module urllib, on télécharge le fichier zip grâce à l'url contenu dans le fichier de config et on le stock dans Recu avec le nom retrieved.zip.

Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur validRec

extraction_comparaison :

On informe qu'on commence cette étape avec le logger.
On vérifie avec le module zip que le dumb SQL est bien présente dans le fichier zip.
Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur validDump.

Si l'utilisateur a indiqué dans le fichier conf qu'il s'agit de la première fois qu'il lance le script, on saute l'étape de comparaison (car pas de fichier de référence),on change en ***non*** la valeur de **[première_fois]** dans le fichier config, on extrait le dump et on modifie sa date de modification grâce aux module dateTime et os afin qu'elle demeure celle qu'elle avait lors du téléchargement (en effet lorsqu'on extrait un fichier d'un zip ca date de modification est modifié et devient celle actuelle, on résout ainsi ce problème). Toutefois on indique avec validPasPareil que l'on n'a pas effectué la comparaison (triangle jaune :warning:).

S'il ne s'agit pas de la première fois, on compare la date de modification du dump dans le zip avec la date du dump de référence en les mettant dans le même format avec strptime. Si les dates sont similaires ou si la date du dump dans le fichier zip est plus ancienne que celle de référence on informe qu'il est inutile de continuer. On

arrête le script. Sinon si le nouveau dump a été modifié plus récemment que notre référence, on supprime ce dernier et le nouveau dump est extrait du zip et devient notre référence.

Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur validPasPareil. Enfin on supprime le fichier zip.

compression :
On informe qu'on commence cette étape avec le logger.
On stock la date d'aujourd'hui (format : AAAADDMM) qui sera le nom de notre nouvelle archive. On créer cette archive tar.gz puis on y ajoute notre dump pour la sauvegarde.
Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur validCompression.

envoi_smb :
On utilise le module SMBConnection pour se connecter à notre serveur avec les différentes options insérer par l'utilisateur dans le fichier de config. On affiche les différents fichiers présents sur le serveur.
Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur validConn.

Si l'historisation est définie comme active on vérifie que les dates des fichiers du serveur ne dépassent pas le ***nbrJours*** si c'est le cas on les suppriment.

Puis on envoi la nouvelle archive sur le serveur.
Si tout s'est bien passé, on valide cette étape en modifiant l'indicateur validUpload.

La classe envoi :

__init__ :

Prend en paramètre self l'instance de la classe ainsi que les mêmes logger et configuration de la classe script. On initialise les indicateurs de la classe envoi aux mêmes valeurs que ceux de l'objet script.

notif_mattermost :

En utilisant le module requests on créer un message en format JSON contenant les différents indicateurs des étapes du script qui soit sont sous la forme validé (white_check_mark), problème (no_entry) ou pas effectué (warning dans le cas de la première fois). Puis on envoi ce message au salon lié au webhook fourni par l'utilisateur dans le fichier de config.

Envoi_email :

On créer un message sous forme html avec les indicateurs et leur symbole équivalent en html. Puis en utilisant SMTP on va se connecter à l'adresse email source avec son mot de passe pour envoyer à l'adresse email destination le message ainsi que le fichier de log en pièce jointe.

Le Main :

Nous créons un objet de classe script prenant en paramètres le fichier de log ainsi que le fichier de config.

On exécute toutes les méthodes de la classe une par une en vérifiant à chaque fois qu'il n'y a pas eu de problème (validé).

Une fois toutes les étapes passées on créer un objet informe de la classe envoi grâce à l'importation du fichier inform.py. On passe en paramètres tous les indicateurs d'étapes du script ainsi que logger, ainsi que la configuration de notre script .
Nous terminons par utiliser les méthodes notif_mattermost() et envoi_email() pour faire part des résultats et terminer le script.

4 – Exécution automatique

Pour lancer automatique et de manière régulière notre script nous utilisons le programme Planificateur de tâches de Windows.

Nous avons choisi ce procédé pour deux raisons : Nous avons travailler sur un environnement Windows et aussi car nous connaissions déjà ce programme de part nos précédentes expériences.

Il permet de créer des « tâches » qui se réaliseront de manière régulière et avec un déclencheur personnalisable que ce soit en fonction d'une certaine fréquence (journalière, hebdomadaire…) ou lors d'un évènement particulier.

Le principe est d'exécuter notre script python directement avec l'exécutable python.exe que l'on a installé ou en utilisant un script batch (.bat) qui lui-même lancera cet exécutable, il est aussi possible d'exécuter manuellement le fichier .bat en double-cliquant dessus (sans passer par le planificateur de tache) pour vérifier son fonctionnement.

Pour plus amples informations, les tutoriels sont disponibles dans la doc utilisateur