FISE 2

Mémoire technique

SCRIPTING SYSTEM

Kim-Celine Franot
2021 - 2022

Table des matières

Organisation du code

Pour réaliser ce projet, j'ai décidé de séparer mon code en 5 classes. Chaque classe est responsable d'une partie du fonctionnement de l'utilitaire. Les classes Zip et SMBServer effectuent les fonctions principales de l'utilitaire c'est-à-dire récupérer le fichier ZIP, extraire son contenu, vérifier son contenu et le déposer sur un serveur SMB distant sous forme d'une archive TAR. La vérification du serveur SMB est également effectuée par la classe SMBServer.

Les classes Log, Email et Mattermost sont considérées comme des classes annexes car elles ne jouent aucun rôle dans le bon fonctionnement de l'utilitaire. En effet ces classes permettent juste d'informer l'utilisateur du fonctionnement de l'utilitaire à travers un fichier de logs, un envoi de notification ou encore un envoi de mail.

Une description détaillée des classes est effectuée dans la partie Description des classes.

Principe de fonctionnement

Au lancement de l'utilitaire, un objet de type Zip est instancié avec le chemin vers le fichier de configuration passé en paramètre. L'objet appelle ensuite la méthode **get_config()** pour récupérer l'ensemble des valeurs saisies par l'utilisateur dans le fichier de configuration. La méthode **download_unzip()** est ensuite appelée pour récupérer le fichier ZIP. La méthode **check_file_version()** est appelée pour vérifier si le dump SQL est différent.

Si la valeur est vraie, le fichier SQL est compressé dans une archive par la méthode **compress_targz()**. Un objet de type SMBServer est crée et un appel des méthodes **connect()**, **send_file()**, **check_file()**, **check_server()**, et **disconnect()** est effectué pour déposer l'archive TAR sur le serveur distant, vérifier sa présence après le transfert et vérifier la validité des autres archives avant de fermer la connexion.

Dans le cas contraire, on se connecte au serveur distant avec **connect()** et on vérifie les autres fichiers sur le serveur avec **check_server()** avant de fermer la connexion.

Enfin, la méthode **send_notification_email()** de la classe Log est appelée pour envoyer les notifications et/ou les mails si demandés avant d'appeler la méthode **clean_directory()** pour enlever les divers éléments du répertoire de travail.

Description des classes

Zip

Cette classe constitue la première partie de l'utilitaire. Elle va d'abord récupérer l'ensemble des données saisies dans le fichier de configuration. Les différentes méthodes permettent à cette classe de télécharger le fichier ZIP contenant le dump SQL à partir de l'URL fourni dans le fichier de configuration. Le dump est ensuite extrait du fichier ZIP et est comparé à une ancienne version pour déterminer si son contenu est différent ou non. S'il est différent, le fichier SQL est compressé dans une archive TAR dans le dossier de travail.

Cette archive sera ensuite traitée par la classe Samba pour la transférer sur le serveur SMB distant.

Cette classe nécessite d'importer os, tarfile, io.BytesIO, hashlib, datetime, json, zipfile, requests et la classe Log.

Cette classe est composée de 7 attributs et de 5 méthodes.

Attributs :

config: String, nom du fichier de configuration (passé en paramètre)

name_tgz: String, nom de l'archive TAR qui correspond à la date actuelle sous le format YYYYDDMM.tgz

prev_sql_file: String, nom de l'ancien fichier SQL initialisé à 'prev_dump.sql'

url_zip : String, url pour télécharger le fichier ZIP (inscrit dans le fichier de configuration)

name_zip: String, nom du fichier ZIP (extrait à partir de l'url)

name_sql_file: String, nom du dump SQL se trouvant dans le fichier ZIP (inscrit dans le fichier de configuration)

log: Log, objet de type Log utilisé pour inscrire les logs pour les différentes actions effectuées

Méthodes :

get_config(): None

Cette méthode permet de récupérer les valeurs saisies par l'utilisateur dans le fichier de configuration concernant l'url pour le fichier zip, le nom du dump SQL et les valeurs pour l'envoi de notifications et de mails. Pour cela, le fichier de configuration est ouvert en lecture et on utilise la méthode **load()** de la librairie json (puisqu'il s'agit d'un fichier JSON) pour extraire l'ensemble des données. Les attributs sont alors instanciés avec celles du fichier.

download_unzip(): None

Cette méthode permet de récupérer le fichier ZIP à partir de l'adresse URL renseignée et de vérifier la présence du dump SQL dans l'archive et de l'extraire vers le répertoire de travail.

Pour cela, la méthode **get()** de la librairie requests permet de faire une requête sur l'URL . Un objet de type zipfile va permettre de récupérer le contenu du fichier ZIP se trouvant à l'adresse URL. On vérifie ensuite que l'objet de type zipfile contient un fichier qui correspond à celui du dump SQL renseigné par l'utilisateur avec la méthode **namelist()** de la librairie zipfile. Si le dump SQL est bien présent dans l'archive ZIP, on l'extrait dans le répertoire de travail avec la méthode **extract()** et on ferme la lecture de l'objet de type zipfile avec la méthode **close().**

check_file_version(): boolean

Cette méthode permet de vérifier si le contenu du fichier SQL est différent d'une ancienne version.

Pour cela, on vérifie la présence d'une ancienne version du fichier SQL nommé 'prev_dump.sql' dans le répertoire de travail. Si aucun fichier correspond, cela signifie qu'il n'y a pas d'ancienne version et que le fichier SQL actuel est le premier dump depuis le fonctionnement de l'utilitaire. La valeur True est retournée.

Dans le cas contraire, on va faire un hachage du contenu de l'ancienne version et de la nouvelle version avec la méthode **sha1()** de la libraire hashlib et comparer le résultat final avec la méthode **hexdigest**().

Si le hachage est identique pour les 2 fichiers, cela signifie qu'il n'y a aucune différence entre l'ancien dump et le nouveau. Le nouveau fichier ne va donc pas être compressé dans une archive et est supprimé du répertoire de travail. La valeur False est retournée. Si le hachage est différent cela signifie que le contenu a évolué entre l'ancienne version et la nouvelle. La valeur True est retournée.

compress_targz(): None

Cette méthode permet de compresser le fichier SQL dans une archive TAR.

Pour cela, on ouvre en écriture une archive avec la méthode **open()** de la librairie tarfile. On ajoute ensuite le fichier SQL dans l'archive avec la méthode **add()**. Une fois ceci effectué, on supprime du répertoire de travail l'ancien fichier SQL et on renomme le nouveau fichier par 'prev_dump.sql'.

clean_directory(): None

Cette méthode permet de nettoyer le répertoire de travail en supprimant l'archive TAR une fois que l'utilitaire a été exécuté.

SMBServer

Cette classe gère la seconde partie du fonctionnement de l'utilitaire. Elle va d'abord récupérer l'ensemble des valeurs du fichier de configuration concernant le serveur distant SMB de la même manière que la classe Zip.

La classe SMBServer permet par la suite de se connecter sur le serveur distant à partir des données renseignées dans le fichier de configuration, de transférer l'archive TAR vers le serveur distant, de vérifier que le fichier a bien été transféré et de vérifier la validité de l'ensemble des fichiers qui sont stockés sur le serveur distant.

Cette classe nécessite d'importer datetime, json, smb.base et smb.SMBConnection.

Cette classe est constituée de 10 attributs et de 5 méthodes.

Attributs :

log: Log, objet de type Log pour la gestion des logs (passé en paramètre)

file: String, nom de l'archive TAR à stocker sur le serveur distant (passé en paramètre)

ip: String, adresse ip du serveur distant (inscrit dans le fichier de configuration)

hostname: String, nom de la machine où se trouve le serveur SMB (inscrit dans le fichier de configuration)

username: String, nom de l'utilisateur Samba (inscrit dans le fichier de configuration)

password: String, mot de passe associé à l'utilisateur Samba (inscrit dans le fichier de configuration)

save_days: String, durée de conservation des archives sur le serveur SMB

sharename: String, nom du partage SMB

path: String, chemin vers le répertoire de stockage des archives sur le serveur SMB

samba: SMBConnection,

Méthodes :

connect(): None

Cette méthode permet d'ouvrir une connexion au serveur SMB distant avec les attributs **username**, **password** et **hostname**. La méthode **connect()** de la librairie SMBConnection est appelée pour gérer la connexion.

disconnect(): None

Cette méthode permet de fermer la connexion au serveur distant en appelant la méthode **close()** de la librairie SMBConnection.

send_file(): None

Cette méthode permet de transférer l'archive TAR vers le serveur distant si une connexion a été ouverte. Pour cela, l'archive est ouverte en écriture avec la méthode **open()** et on appelle la méthode **storeFile()** de la librairie SMBConnection pour transférer l'archive sur le serveur distant dans le répertoire de destination.

check_server(): None

Cette méthode permet de vérifier la validité des fichiers stockés sur le serveur distant et de les supprimer si nécessaire.

Pour cela, on récupère la liste des éléments se trouvant dans le répertoire distant avec la méthode **listPath().** On calcule la date maximale de stockage à partir de la date actuelle et de la valeur de l'attribut **save_days**. Pour chaque élément de la liste, on vérifie qu'il s'agit d'un fichier et on compare le nom du fichier par rapport à la date maximale. Si le nom correspond à une date inférieure à la date maximale calculée, on enlève le fichier du répertoire avec la méthode **deleteFiles()** de SMBConnection.

check_file(): boolean

Cette méthode permet de vérifier que l'archive a bien été transférée vers le serveur distant. Pour cela, on récupère la liste des éléments se trouvant dans le répertoire de stockage du serveur SMB avec la méthode **listPath()** de la librairie SMBConnection et on itère chaque élément pour voir si un fichier porte le même nom que celui de l'archive qui a été transférée. Si l'archive est présente, la valeur True est retournée, sinon False est retourné.

Log

Cette classe permet de faire la gestion des logs pour chaque tâche effectuée lors du fonctionnement de l'utilitaire. Ils peuvent s'agir de logs d'informations, de warnings ou encore d'erreurs. Si l'utilisateur a demandé un envoi de notification et/ou de mails, cette classe va faire appel aux méthodes des classes Mattermost et/ou Email pour initialiser le contenu à envoyer et pour les envoyer.

Cette classe nécessite d'importer logging, os et les classes Mattermost et Email.

Cette classe est constituée de 6 attributs et de 5 méthodes.

Attributs :

nb_error: int, nombre d'erreurs soulevées lors du fonctionnement de l'utilitaire

nb_warning: int, nombre de warnings soulevés lors du fonctionnement de l'utilitaire

mattermost: Mattermost, objet pour gérer la gestion des notifications

send_notification: string, valeur passée en paramètre dans le constructeur correspondant au choix d'envoi de notification par l'utilisateur dans le fichier de configuration

email: Email, objet pour gérer les mails

send_emails: string, valeur passée en paramètre dans le constructeur correspondant au choix d'envoi de mail par l'utilisateur dans le fichier de configuration

Méthodes :

info(string, string): None

Cette méthode est appelée pour ajouter un log d'information lorsqu'une tâche a été effectuée avec succès. Si l'envoi de notification est configuré, un appel de la méthode **set_info()** de la classe Mattermost est effectué pour ajouter la tâche. De même si un envoi de mail est configuré, un appel de la méthode **add_content()** de la classe Email est effectué pour ajouter la tâche.

error(string, string): None

Cette méthode est appelée pour ajouter un log d'information lorsqu'une erreur a été soulevée lors de la réalisation d'une tâche. Si l'envoi de notification est configuré, un appel de la méthode **set_error()** de la classe Mattermost est effectué pour ajouter la tâche. De même si un envoi de mail est configuré, un appel de la méthode **add_content()** de la classe Email est effectué pour ajouter la tâche. De plus, l'attribut **nb_error** est incrémenté.

warning(string, string): None

Cette méthode est appelée pour ajouter un log d'information lorsqu'un warning a été rencontrée lors du déroulement d'une tâche. Si l'envoi de notification est configuré, un appel de la méthode **set_warning()** de la classe Mattermost est effectué pour ajouter la tâche. De même si un envoi de mail est configuré, un appel de la méthode **add_content()** de la classe Email est effectué pour ajouter la tâche. De plus, l'attribut **nb_warning** est incrémenté.

get_file(): string

Cette méthode permet de retourner le chemin vers le fichier de log. Cette méthode est appelée par la classe Email dans la méthode **attach_file()** si le fichier de log doit être inclus en pièce jointe au mail.

send_notification_email(): None

Cette méthode permet d'appeler les méthodes **send_notification()** de la classe Mattermost si un envoi de notification est demandé et **send_email()** de la classe Email si un envoi de mail est demandé.

Dans le cas de la notification, on vérifie que la valeur de l'attribut **send_notification** correspond à « always » ou « error » et que la valeur de l'attribut **nb_error** est supérieure à 0.

Dans le cas du mail, on vérifie que la valeur de l'attribut **send_emails** vaut « yes » ou « y » et on initialise l'attribut **err_warning** de la classe Email selon le nombre d'erreurs soulevé lors du fonctionnement de l'utilitaire avant d'appeler la méthode **send_email()** de la classe Email.

Email

Cette classe permet de gérer l'envoi de mail vers les destinataires renseignés dans le fichier de configuration. Le mail récapitulera l'ensemble des tâches réalisées lors de l'exécution de l'utilitaire. Le mail pourra intégrer le fichier de log si demandé.

Cette classe nécessite d'importer email, json, smtplib, ssl, datetime et email_validator.

Elle est constituée de 10 attributs et 6 méthodes.

Attributs :

server : smtplib, objet pour la gestion à la connexion au serveur smtp

content : Array, variable qui permet de stocker le contenu des différentes tâches

log : Log, objet pour la gestion des logs

err_warning : string, variable qui contient le titre du mail selon le nombre d'erreurs et/ou de warnings soulevé lors de l'exécution de l'utilitaire.

host : string, nom du serveur smtp utilisé pour l'envoi des mails (inscrit dans le fichier de configuration)

port : string, numéro du port smtp utilisé (inscrit dans le fichier de configuration)

auth : dictionnaire contenant l'email et le mot de passe associé de l'utilisateur (inscrit dans le fichier de configuration)

include_log : string, valeur correspondant au choix de l'utilisateur concernant l'ajout du fichier en pièce jointe (inscrit dans le fichier de configuration)

title : string, objet du mail (inscrit dans le fichier de configuration)

dest : Array, contient les adresse mails des destinataires pour l'envoi des mails (inscrit dans le fichier de configuration)

Méthodes ;

login(self) : None

Cette méthode permet de se connecter au compte de l'adresse mail utilisé pour l'envoi des mails. L'attribut **server** étant un objet de type smtplib fait appel à la fonction **login()** pour se connecter en passant en paramètres les valeurs « email » et « password » contenues dans l'attribut **auth**.

login_and_send(self) : None

Cette méthode permet de se connecter en faisant appel à la méthode **login()** définie précédemment et de créer un objet de type MIMEMultipart. On appel ensuite la méthode **attach_file()** pour inclure le fichier de log si nécessaire et le mail est envoyé au(x) destinataire(s) avec la méthode **sendmail()** de la libraire smtplib.

send_smtp(self) : None

Cette méthode permet d'initialiser d'ouvrir une connexion avec le serveur smtp indiqué dans le fichier de configuration. Une fois la connexion au serveur effectuée, la méthode **login_and_send()** est appelée.

add_content(self, String, String) : None

Cette méthode permet de rajouter une tâche et son message passés en paramètre au tableau de l'attribut **content.** Il s'agit d'une méthode qui est appelée par la classe Log au moment de l'ajout des différents logs.

attach_file(self, MIMEMultipart) : MIMEMultipart

Cette méthode permet d'inclure le fichier de log en pièce jointe. Pour cela, la valeur de l'attribut **include_log** est inspectée et si elle correspond à « yes » ou « y », la méthode **get_file()** de la classe Log est appelé pour obtenir le chemin vers le fichier de log. Le fichier de log est alors ouvert en écriture et ajouter au mail avec la méthode **attach()** de la librairie email. Le message est ensuite retourné.

format_mail(self) : String

Cette méthode permet d'ajouter au corps du mail le contenu du tableau de l'attribut **content** ligne par ligne. Le contenu est ensuite retourné.

Mattermost

Cette classe permet de gérer l'envoi de notification vers un serveur Mattermost. La notification envoyée est sous forme de tableau et permet de récapituler l'ensemble des tâches effectuées et leur statut d'exécution.

Cette classe nécessite d'importer datetime, requests.

Elle est constituée de 3 attributs et de 5 méthodes.

Attributs :

log : un objet de la classe Log qui permettra de rajouter des logs en cas d'erreur lors de l'envoi de notification.

webhook : un string contenant l'url du webhook pour le serveur Mattermost

mattermost_content : un dictionnaire constitué de 2 clés : « tasks » pour ranger les intitulés des différentes tâches effectuées ; « status » pour ranger les codes pour les pictogrammes à afficher associés à chaque tâche.

Méthodes :

set_info(self, task) : None

Cette méthode permet d'ajouter le titre de la tâche (passé en paramètre) dans le dictionnaire **mattermost_content** à la clé « tasks » et d'ajouter à la clé « status » le code pour un pictogramme de réussite.

set_warning(self, task) : None

Cette méthode permet d'ajouter le titre de la tâche (passé en paramètre) dans le dictionnaire **mattermost_content** à la clé « tasks » et d'ajouter à la clé « status » le code pour un pictogramme de warning.

set_error(self, task) : None

Cette méthode permet d'ajouter le titre de la tâche (passé en paramètre) dans le dictionnaire **mattermost_content** à la clé « tasks » et d'ajouter à la clé « status » le code pour un pictogramme d'erreur.

format_notification(self) : String

Cette méthode permet de formatter le contenu de la notification à envoyer sous la forme d'un tableau. La date actuelle est d'abord récupérée pour être insérée dans un titre qui va être stocké dans une variable de type String.

Pour chaque ligne de l'attribut **mattermost_content**, on va ajouter le contenu des clés « tasks » et « status » à la variable de type String.

Enfin, la variable de type String est retournée.

send_notification(self) : None

Cette méthode permet de récupérer le contenu de la notification en faisant appel à la méthode **format_notification()** et utilise la méthode **post()** de la librairie **requests** afin d'envoyer la notification à l'url renseigner dans l'attribut **webhook**.