MÉMOIRE TECHNIQUE

ARCHIVE MANAGER - L'UTILITAIRE DE GESTION D'ARCHIVAGE

telecom saint-étienne
école d'ingénieurs
nouvelles technologies

MAXIME PERRIN

SOMMAIRE

INTRODUCTION

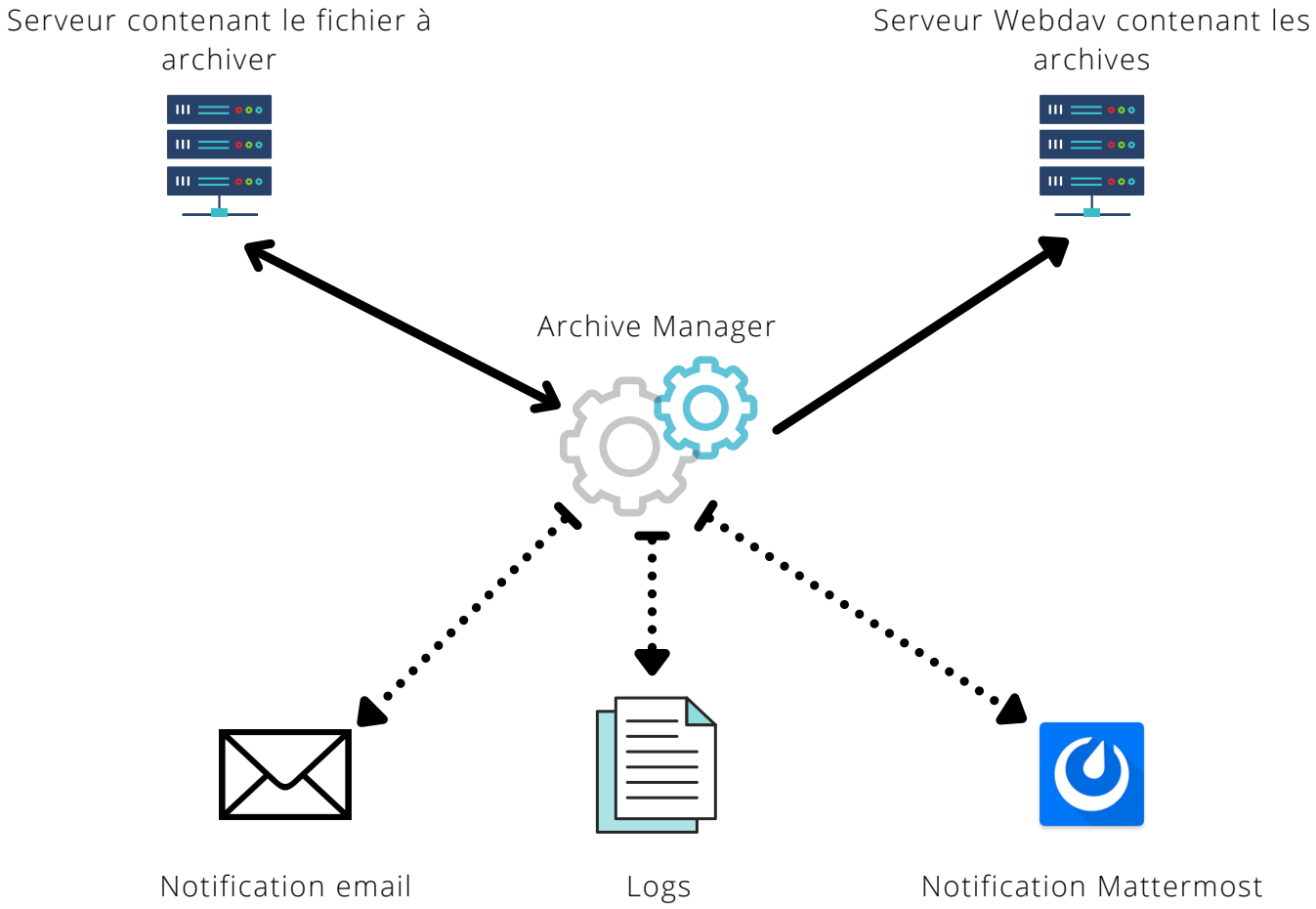
Ce mémoire répertorie les choix techniques ainsi que des explications concernant les différentes parties du projet. Celui-ci a été réalisé dans le cadre du cours *Scripting System*.

Le but du projet est de créer un script permettant d'automatiser l'archivage d'un fichier compressé obtenu à partir d'un serveur distant.
L'utilitaire vérifie si le fichier n'est pas le même que celui stocké en local et si tout est bon, il l'envoie sur un serveur Webdav.

Le projet est disponible sur Github à cette adresse : https://github.com/maximeperrindev/scripting-system.
Le guide d'utilisation et d'installation est disponible sur le repository dans le fichier README.md. Cette notice est rédigée en anglais afin de permettre à tous le monde de pouvoir l'utiliser.

Archive Manager a été créé dans le but de répondre à une problématique d'optimisation et de gain de temps. C'est pourquoi l'architecture est simple et l'utilitaire est conçu pour que l'administrateur ait le moins de chose à faire.

FONCTIONNEMENT



Dans un premier temps, *Archive Manager* va récupérer le fichier compressé à archiver. Ensuite, il va vérifier que tout est bon en local :

- Dump SQL existant
- Fichier différent que le dernier téléchargé

Si tout est bon, on extrait le dump et on crée une archive (.tgz)
On établit une connexion avec le serveur Webdav et on vérifie :

- Connexion bien établie
- Répertoire 'archive' existant sinon on le crée

Encore une fois, si tout est vérifié, on upload le fichier .tgz sur le serveur
On va également vérifier si des fichiers sont "périmés" avec les paramètres donnés dans le fichier de configuration.
Chaque résultat d'action est répertorié dans un fichier de log afin de pouvoir mieux suivre le fonctionnement et comprendre les éventuelles erreurs.

FONCTIONNEMENT

Une fois que la tâche est exécutée et terminée (avec succès ou non), on envoi des notifications :

- Par mail (liste configurable) :
  - En cas d'erreur, le mail indique qu'il y a eu un problème et on trouve en pièce jointe le fichier de log.
  - En cas de succès, le mail indique que tout va bien et que le script s'est bien exécuté.

- Par Mattermost (webhook configurable) :
  - En cas d'erreur, on peut retrouver ce tableau :

Archiving report for 21 December, 2021

Please, check logs if some errors occured.

Task	State
Get config data	✅
Download zip	✅
Zip extracted	⚠️
Different file	⚠️
Check for outdated files	✅
Archive uploaded	⚠️

  - En cas de succès, on peut retrouver ce tableau :

Archiving report for 21 December, 2021

Please, check logs if some errors occured.

Task	State
Get config data	✅
Download zip	✅
Zip extracted	✅
Different file	✅
Check for outdated files	✅
Archive uploaded	✅

ORGANISATION DU CODE

Pour développer *Archive Manager*, il a fallut découper fonctionnellement le programme afin qu'il soit facilement modifiable. Bien entendu, le découpage en fonction ne se fait que si cela est justifié (pour utiliser des processus qui se répètent dans le programme par exemple).

On peut noter plusieurs fonctions :

- send_success_mail qui envoie un mail de succès
- send_error_mail qui envoie un mail d'erreur avec les logs
- make_tarfile qui crée un fichier compressé .tgz
- hash_file qui retourne le hash SHA-1 d'un fichier passé en paramètre
- hash_zip fait comme hash_file mais avec un fichier zip
- downloadFileFromUrl télécharge un fichier à partir d'un url

On récupère également le fichier de configuration via une bibliothèque qui gère les fichiers .ini

```
[FILE]
URL = URL_TO_FILE
SQLFILE = NAME_OF_DUMP_FILE_WITH_EXTENSION
[SERVER]
URL = URL_TO_WEBDAV_SERVER_WITH_/webdav
USERNAME = USERNAME_OF_WEBDAV_SESSION
PASSWORD = USERNAME_OF_WEBDAV_SESSION
CONSERVATION_TIME = NUMBER_OF_DAYS
[MAIL]
ENABLED = TRUE
USERNAME = USERNAME(MAIL ADDRESS)
PASSWORD = PASSWORDFORMAIL
ATTACH_LOG = TRUE
MAIL_LIST = MAIL LIST WITH SPACES
[MATTERMOST]
WEBHOOK = WEBHOOK_URL
```

conf.ini

```
## GET CONFIG
try:
    config = configparser.ConfigParser()
    config.read('conf.ini')
    file_conf = config['FILE']
    server_conf = config['SERVER']
    mail_conf = config['MAIL']
    mattermost_conf = config['MATTERMOST']
    confSuccess = True
except:
    confSuccess = False
```

Récupération de la configuration

ORGANISATION DU CODE

Pour développer l'utilitaire, il a fallut utiliser de nombreuses variables pour simplifier le code. Voici le détail :

Variables permettant de connaître l'état (succès ou non) des différents processus :

Variable (boolean)	Exemple de valeur
confSuccess	True / False
downloadZipSuccess	True / False
zipExtractedSuccess	True / False
diffDateSuccess	True / False
sentToServerSuccess	True / False
checkOutdated	True / False
success	True / False

Variables issues du fichier de configuration :

Variable (List)	Exemple de valeur

config	[
	« SERVER » :{
	« URL » : http://169.254.8.10/webdav
	}
	...
	]

File_conf	Config[« FILE »] => Liste des paramètres du .ini
Server_conf	Config[« SERVER »] => Liste des paramètres du .ini
Mail_conf	Config[« MAIL »] => Liste des paramètres du .ini
Mattermost_conf	Config[« MATTERMOST »] => Liste des paramètres du .ini

Variables qui stockent le sigle à afficher dans le tableau Mattermost :

Variable (String)	Exemple de valeur
confSuccessMessage	:white_check_mark : / :warning :
downloadSuccessMessage	:white_check_mark : / :warning :
extractedSuccessMessage	:white_check_mark : / :warning :
diffSuccessMessage	:white_check_mark : / :warning :
sentSuccessMessage	:white_check_mark : / :warning :
outdatedMessage	:white_check_mark : / :warning :

Le logger (objet qui permet d'écrire dans le fichier de log) :

Variable (logger)	Exemple de valeur
logger	Pas de valeur mais appel via logger.info(log) / logger.error(log)

ORGANISATION DU CODE

Toutes les autres variables utilisées dans le code principal et dans les fonctions :

Variable (type)	Exemple de valeur
Zip_ref (zipfile)	./resources/test_export.zip
listOfFileNames	['test_export.sql','__MACOSX/._test_export.sql']
nbDownload (int)	0/1/2....
files (list[string])	['archive/20211221.tgz']
expiration (datetime)	2021-12-23 00 :00 :00
fileName (Path)	Archive_file 20212612.tgz
Date_file (datetime)	2021-12-26 00 :00 :00
client (Webdav Client)	Webdav client with all methods related to
same (boolean)	TRUE / FALSE
new_filename (string)	20212612.tgz
today (datetime)	2021-12-26 00 :00 :00
Mail_list (list[string])	[mail1@domain.ext, mail2@domain.ext , mail3@domain.ext]
Mail(String)	Mail1@domain.ext
subject (string)	"#### Archiving report for "+ today+"\nPlease, check logs if some errors occured.\n"
table (string)	| Task | State |\n"+"|:----------|:----------|\n"+"| Get config data | "+ confSuccessMessage +"\n" +"| Download zip | "+ downloadSuccessMessage+"\n" +"| Zip extracted | "+ extractedSuccessMessage+"\n" +"| Different file | "+ diffSuccessMessage+"\n"+"| Check for outdated files | "+ outdatedMessage+"\n" +"| Archive uploaded | "+sentSuccessMessage
headers (dict[string, string])	{'Content-type' : 'appliction/json'}
data (list[string])	{"username": "Archive Manager","icon_url": "https://mattermost.org/wp-content/uploads/2016/04/icon.png","text": subject + "\n" + table}
x (Requests)	Request response value (body, headers, status code...)
Msg(MIMEMultipart)	Mail body message
Mailserver(smtpMailServer)	Mail server object (connected with smtp protocol)
h (hash object)	object for sha1 hash
Chunk (int)	1024 (chunk size for hashing)

CHOIX TECHNIQUES

Il y avait de nombreuses façon de faire pour réaliser ce projet. Il a donc fallu faire des choix techniques.

- **Fichier .ini**

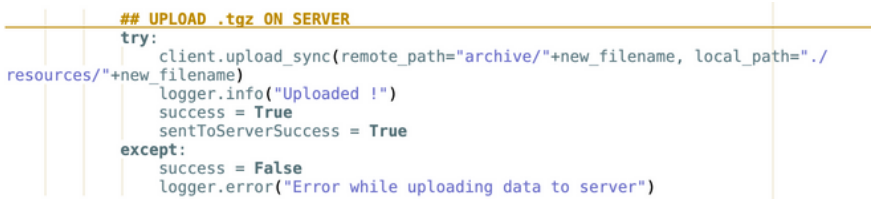
Pour configurer notre utilitaire, nous avons besoin d'utiliser un fichier de configuration. Il en existe une multitude comme par exemple JSON, ini, ou yml.
Archive Manager utilise un fichier .ini car c'est un format très connu et beaucoup utilisé dans les systèmes d'exploitations (Windows, Unix) donc cela s'adapte bien à un gestionnaire de fichier.

- **Serveur WEBDAV**

C'est un serveur qui permet d'héberger une gestion de fichier sous un protocole HTTP. Il est assez ancien mais il a été retenu pour sa puissance et sa pertinence avec le projet. En effet, une connexion via un protocole HTTP est un peu moins lourde et plus simple qu'une connexion par accès SSH. De plus, ce type de serveur est adapté à la saisie "collaborative" ce qui pourrait permettre à un autre script de travailler de concert avec *Archive Manager*. Puisqu'il est intéressant de se projeter plus loin que les limites de ce projet, il faut anticiper certaines problématiques comme l'édition simultanée d'un fichier. On pourrait envisager par la suite de mettre à jour ce serveur avec une infrastructure de type Cloud pour moderniser le fonctionnement.

Le code est découpé pour chaque fonction du cahier des charges. En effet, il a été découpé par des commentaires afin d'identifier quelle partie était associée à chaque portion de code.
Presque aucune valeur "en dur" n'est utilisée afin de maximiser la personnalisation de l'utilitaire.

Chaque opération pouvant entraîner une erreur est entourée avec des "try...except". Cela permet de gérer les exceptions et donc d'adapter le programme à toute éventualité.



```
            ## UPLOAD .tgz ON SERVER
            try:
                client.upload_sync(remote_path="archive/"+new_filename, local_path="./
resources/"+new_filename)
                logger.info("Uploaded !")
                success = True
                sentToServerSuccess = True
            except:
                success = False
                logger.error("Error while uploading data to server")
```

CONCLUSION

Pour conclure, *Archive Manager* a permis de voir et comprendre comment gérer efficacement un utilitaire en Python. La communication avec un serveur et rendre son code adaptable facilement a été tout particulièrement intéressant. Car, au delà de l'application technique, cela permet de comprendre des concepts essentiels dans le monde de la programmation et de la gestion de projet.
Le programme a vraiment été pensé pour être réutilisable et modifiable. La notice d'utilisation a d'ailleurs été rédigée en Markdown. C'est une syntaxe très utilisée dans le monde du développement informatique pour documenter son code.

Pour faire un retour critique sur le projet, il aurait pu être intéressant de découper les fonctionnalités en plusieurs fichiers ou "modules". On peut également penser à la suite du projet en suggérant de créer un script côté serveur (WEBDAV) traitant automatiquement le fichier reçu et exécutant quelques actions.